A SOFTWARE ENGINEERING APPROACH TO DEVELOPING AN OBJECT-ORIENTED LEXICAL ACCESS DATABASE AND SEMANTIC REASONING MODULE

by

Wendy Mair Zickus

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences

Spring 1995

A SOFTWARE ENGINEERING APPROACH TO DEVELOPING AN OBJECT-ORIENTED LEXICAL ACCESS DATABASE AND SEMANTIC REASONING MODULE

by

Wendy Mair Zickus

Approved: _____

Patrick W. Demasco, MS

Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Kathleen F. McCoy, Ph.D.

Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Errol Lloyd, Ph.D.

Chairman of the Department of Computer and Information Sciences

Approved: _____

Carol E. Hoffecker, Ph.D.

Associate Provost for Graduate Studies

ACKNOWLEDGMENTS

This work has been supported by a Rehabilitation Engineering Research Center Grant from the National Institute on Disability and Rehabilitation Research of the U.S. Department of Education (#H133E30010). Additional support has been provided by the Nemours Research Programs.

I would like to thank my mother, Mary G. Mair, for being a surrogate mother to my daughter, Rebecca, while I was working on my graduate studies. I would also like to thank both her and my father, Robert D. Mair, for being supportive of my work over the past few years.

To my advisors, Kathleen F. McCoy and Patrick W. Demasco, I am indebted for their guidance, patience, and faith in me. When I started working with them in the Natural Language Processing Lab at the Center for Applied Science and Engineering, located at the A. I. duPont Institute, I was a new graduate student in the Computer Science Department interested in Object-Oriented Programming. Now I am an Object-Oriented Natural Language Processing programmer with some expertise in Computational Lexical Semantics.

I want to thank my daughter, Rebecca, for understanding all of the times when I could not be there due to projects and academic obligations, and especially for allowing me to help her celebrate her birthday in June instead of May due to finals.

And last, but definitely not least, I'd like to thank my husband, Timothy E. Zickus for his love, support and seemingly endless editing of this thesis. His pride and belief in my abilities and capabilities were of immeasurable importance to the completion of my research work at The University of Delaware.

TABLE OF CONTENTS

LIST OF FIGURES ..... **viii**
ABSTRACT ..... **ix**

Chapter

1 INTRODUCTION **1**

2 RELATED RESEARCH **7**
Motivation 7
Application Domain: Augmentative and Alternative Communication 8
AAC Language Technology Overview 9
Some Natural Language Processing (NLP) used in AAC Applications 10
Need for Word Knowledge in NLP based AAC Systems 11
Compansion 13
Computational Lexical Semantic Overview 17
Dictionaries and Lexicons 20
Paper-Based 20
On-Line 21
WordNet 22
Efforts in Merging Lexical Resources 30

3 LAD DESIGN **34**
Motivation 34
Functional Interface 36
Argument Structure 36
Functions for Semantic Knowledge 37
Functions for Syntactic Knowledge 44
Functions for Other Specialized Knowledge 45

LAD Resources 46
WordNet 47
Case Frames 49
Morphology 49
Phonetic Information 50
Syllabification Information 50
Frequency Information 51
LAD Mapping 51
Semantic Categories 52
Queries Requiring Multiple Databases 53
Complicated Queries 54
Application Areas 55
Object-Oriented Design 56
Extensibility 59
Implementation Status 60

4 SEMANTIC REASONING MODULE 61
Motivation 61
Case Frames and Semantic Reasoning 62
Semantic Categories 63
Preferences 63
SRM and LAD (Knowledge Representations) 67
Secondary Verb Frames vs. WordNet Verb Frames 67
Semantic Output 68
SRM Processing 69

5 CONCLUSIONS 73
Accomplishments 73
LAD Implementation Status 74
WordNet Usage 75
SRM and LAD Results 75
SRM Performance vs. Compansion Performance 77
Discussion 80
Future Work 81

BIBLIOGRAPHY 83

APPENDIX A LAD SOURCE DESCRIPTION 90

APPENDIX B SRM SOURCE DESCRIPTION 98

APPENDIX C SEMANTIC CATEGORIES 108

APPENDIX D WORDNET VERB FRAMES 111

LIST OF FIGURES

Figure 2.1	Specified Case Frame for the Verb BREAK	15
Figure 2.2	List of 25 Unique Beginners for WordNet Nouns	23
Figure 2.3	Hyponymic Relations of Seven WordNet Unique Beginners	24
Figure 2.4	WordNet Verb Sentence Frames	28
Figure 3.1	Language Access Database Diagram	48
Figure 3.2	LAD Object-Oriented Class Hierarchy	58
Figure 4.1	Specified Case Frame for the Verb LIKE	66
Figure 4.2	List of Filled Case Frames	69
Figure 4.3	Algorithm for the Generation of Filled Case Frames (Part 1)	70
Figure 4.4	Algorithm for the Generation of Filled Case Frames (Part 2)	71

ABSTRACT

The concept of enabling computers to understand and generate 'natural' language has been enticing to mankind ever since we first saw and heard HAL in the movie "2001: A Space Odyssey." However, the state of the art in Natural Language Processing (NLP) is a long way from creating a reality of this concept. One of the major bottlenecks in the implementation of NLP is the lexicon: the place where the system's information about words is stored. There are difficulties in deciding what information should be stored in a lexicon, and even greater difficulties in acquiring this information. To date there have been several efforts which provide on-line lexical database resources with varying amounts of lexical information for NLP systems. One problem that remains, however, is that a particular NLP application may need information from several such sources, and there is no standard way to access and combine information from several lexical resources.

This thesis describes the Language Access Database (LAD) system, a system designed to incorporate multiple lexical databases and tools under one consistent functional interface in order to facilitate systems requiring syntactic, semantic and lexical knowledge. The technical aspects of the design were mainly influenced by research in the areas of Computational Lexical Semantics, Augmentative and Alternative Communication, and Natural Language Processing. The implementational aspects of the design were influenced

by the paradigm of Object-Oriented programming to create a system that is easily extendable and upgradable.

This thesis presents LAD and the Semantic Reasoning Module (SRM), a semantic parsing system designed to test the LAD system's functionality.

Chapter 1

INTRODUCTION

In order for communication to exist, there must be some agreed upon representation of symbols, whether iconic, gestural, vocal, or written, and an agreed upon set of rules upon which to build combinations of symbols into sentences of meaning. Human beings have the capacity to learn how to communicate, to pass along the semantic and syntactic rules to their following generations, and eventually to expand upon and refine these symbols and rules. Through these representations, mankind has the ability to communicate, to generate new ideas and thoughts, to learn and to build upon the knowledge built up in the complex structure called language. According to the American Heritage Dictionary (1985), language is:

> The use by human beings of voice sounds, and often of written symbols that represent these sounds, in organized combinations and patterns to express and communicate thoughts and feelings. A system of words formed from such combinations and patterns, used by the people of a particular country or by a group of people with a shared history or set of traditions.

Since the advent of computers, humans have been searching for ways to improve them. We strive to make them faster, increase their memory capacities, improve the user-interface peripherals and the methods we use to communicate with them. From a business perspective, there is a feeling of frustration at times, voiced about the computer not living

up to its heralded increased productivity and decreased cost expectations. This frustration is partially caused by the high learning curve for employees to master the computer and its software, and partially due to the unnatural communication methods we use to describe our problem to the computer, solve it, and interpret the results. This is one impetus pushing computer scientists to give our programs the capability to understand or generate language in order to provide a more 'natural' way for computer users to be able to communicate with their systems and vice versa.

Natural Language Processing (NLP) is the area within computer science that deals with trying to create ways to process human language for various reasons and applications. In order to do this, there must be some means of representing language structure, and a mechanism for using this representation for generating and understanding language within the computer. Efforts to find solutions to these tasks have focused on three major areas of research: syntactic, semantic and pragmatic. The syntax of a language refers to the grammatical rules defining allowable ways to link words together into sentences. Semantics refers to the meanings of individual words and to the meanings formed by combining the words in a sentence.[1] Pragmatics refers to the overall context in which language is used (i.e., the situational context and past conversational experiences between conversing partners), creating expectations about future language interactions and context. This thesis will focus on the issues raised by the semantics of language.

---

1. One NLP solution used to resolve word sense ambiguities is to look at the individual words in a sentence in relation to the other words of the sentence (i.e., the words in a sentence constrain the possible meanings of each other) (Small & Rieger, 1982).

One of the difficulties faced by the area of semantics is that many words have multiple meanings (e.g., *shoe* can refer to protective covering worn on the foot, a restraining device used to help cars brake, or a container for cards in a casino). Once this is determined, a semantic system must reason about how individual word meanings can be used in resolving the meaning of a sentence. This has caused a focus on the kinds of information needed in order to do semantic parsing and reasoning. Do we need hierarchically linked relationships between words as well as syntactic and definitional information? How should we address the word sense disambiguation issue?[2] Some of this research has resulted in the specialized field of Computational Lexical Semantics, while other research has led to an increased understanding and improved methods and techniques necessary to handle the complex task of understanding and generating natural language.[3]

Augmentative and Alternative Communication (AAC) research focuses on developing technologies to aid in spoken and written communication processes for people with motor and/or cognitive impairments. The abilities of people using AAC devices cover a broad spectrum of capabilities, from the mild to the more severely impaired individuals and from those with a single disability to those with a combination of cognitive, speech or language impairments. AAC devices are designed to enhance the language capabilities of this group and thus come in a variety of forms with a variety of technological options.

---

2. This is one of the major questions faced today by researchers in Computational Semantics. It was the focus of the Post-COLING94 Workshop on Directions of Lexical Research, and is one of the main foci of the upcoming 1st Annual workshop for the IFIP Working Group for Natural Language Processing and Knowledge Representation to be held at the University of Pennsylvania in April 1995.

3. Though the state of the art in NLP still has restricted domains, there is nowadays broader application coverage for NLP systems (Allen, 1994).

Some AAC devices are non-electronic boards or books that may contain letters, words, concepts, or phrases written in traditional orthography or pictorial representations. Others are electronic, containing similar representations and usually accompanied by speech synthesis that provide a user with the capability to select and "speak" an utterance. Some electronic AAC devices include predictive capabilities to allow the user to create messages with a reduced physical load. This thesis will focus on the AAC devices that incorporate Artificial Intelligence and Natural Language Processing techniques and principles. Such systems require knowledge attached to the words in their system dictionaries.

The dictionary requirements for NLP-based AAC systems reflect the needs of the three major NLP research areas: syntactic, semantic and pragmatic. For the syntactic area, there is a need for grammars (i.e., a set of rules that refer to word categories and word patterns), morphological information (e.g., +S for plural, +ING), and word category information (e.g., NOUN, VERB, PREPOSITION). Syntactic knowledge is needed for systems that use word prediction, grammar checkers, and/or language tutoring (McCoy & Demasco, 1995). Semantic knowledge is useful for systems that process and/or generate language. Examples of semantic knowledge needed would include selectional restrictions, case frames, conceptual meaning for words (e.g., the semantic categories for *Mary* are human and animate, *window* are fragile and physical object, *hammer* are tool and physical object), relational knowledge (e.g., a *car* **is-a** vehicle, a *canary* **is-a** bird), attributive knowledge (e.g., a *canary* is small and yellow with wings, feathers and a beak), functional knowledge (e.g., *knives* are used for cutting, *trees* are used for shade, energy and building) and knowledge of parts (e.g., a *car* has a windshield, a hood, brakes, fenders, tires, steer-

ing wheel). Semantic knowledge has proven useful to AAC systems which try to understand the meaning the user of the system is attempting to convey. And finally, for the area of pragmatics, knowledge concerning situational context and communication history, conversational information (i.e., turn-taking rules, organizational patterns for story-telling) is needed. This pragmatic information is important for systems that model typical conversational patterns (Alm et al., 1993).

There is a variety of applications that require large amounts of lexical knowledge. While there are several on-line lexical resources available today, no single one contains all the information which may be necessary for AAC/NLP applications.[4] In the Language Access Database (LAD), we attempt to provide such a resource by creating a virtual dictionary which combines information from several available lexical and linguistic sources in a transparent, seamless manner. This project will provide a uniform dictionary interface to a variety of lexical resources (e.g., WordNet, collocative information, verb case frame data, word frequency data, phonetic information, syntactic information, category information). LAD provides the user (i.e., application program) with a functional interface through which they may pose a variety of queries. The LAD program will search the appropriate lexical resource in order to handle a query. LAD shields the user program from the task of understanding which (set of) lexical resource(s) actually contains the necessary information. Many different kinds of lexical information can be obtained through this consistent user interface. LAD has the capability to extract information needed by systems to enhance their capabilities to understand language, to generate semantically and syntacti-

4. Some of the efforts to develop such on-line lexical resources will be discussed in Chapter 2.

cally correct language, to process language and/or to produce speech-synthesized language, depending on the needs of the individual systems.

The focus of this thesis is the LAD system and its design. Before describing LAD in detail, Chapter 2 provides the technical background. It first motivates the need for a system like LAD by describing the application domain of Augmentative and Alternative Communication concentrating on NLP-based AAC applications. It also covers the area of Computational Lexical Semantics and work in the area of dictionaries (both paper-based and on-line, and efforts at merging lexical resources). The chapter highlights current research, knowledge base needs of systems, and available knowledge that current lexical sources provide. Chapter 3 details the LAD design: its functional interface, lexical resources, mapping capabilities, application areas, object-oriented design, extensibility, and implementation status. This chapter provides a complete description of LAD including the different kinds of lexical information that it can provide, a listing of applications that can benefit from LAD, and the flexibility the LAD design provides with respect to future enhancements and/or additional lexical resources. Chapter 4 details an application written to test and evaluate LAD called the Semantic Reasoning Module. It describes the kinds of information the module needs, its semantic reasoning, and how LAD provides the information needed for it to function properly. Finally, Chapter 5 concludes this thesis with an analysis of the testing and evaluation of LAD, a discussion regarding what LAD does and does not provide, and a look into the future directions of the LAD research project.

Chapter 2

RELATED RESEARCH

Motivation

A great deal of knowledge is required in order to understand language. There are many areas in Computer Science and other professions researching the different 'kinds' of knowledge needed for this task, such as Computational Lexical Semantics, Linguistics, Psycholinguistics, Augmentative and Alternative Communication (AAC), and Natural Language Processing (NLP), just to name a few. Some of these research areas concentrate on the fundamental structure of language, its syntactic and semantic properties, while other areas concentrate on what is needed in order for a computerized system to represent the meaning of textual input to facilitate computer understanding and/or generation of language. They all agree that the task set before them is difficult and may end up being impossible to accomplish in its most complete form. The progress made in this area has already led to vast improvements in computerized understanding and processing within restricted domains and applications (Chen & Huang, 1994). The need for and benefits of systems that will allow for unconstrained input justifies the continued efforts of researchers in this field (McHale & Crowter, 1994). The following section provides an overview of the application area of AAC which was the catalyst of this work. We focus on NLP-based AAC systems and their lexical semantic requirements. The following sections provide

background into the area of Computational Lexical Semantics, on which this thesis has relied for technical information about words (i.e., information that must be associated with words for various kinds of processing). The final section covers research on dictionaries, both paper-based and on-line, for further guidance on the kind of information that should be available on words.

Application Domain: Augmentative and Alternative Communication

The field of Augmentative Alternative Communication (AAC) is devoted to developing alternative technology for people with disabilities[1] that prevent normal means of communication. The technology may provide an alternative means of communication and/or may be used in the assessment and training for these individuals (Demasco et al., 1994). The research has centered around the areas of input (e.g., eye gaze and gesture), language (e.g., representation, organization and processing of language), and output (e.g., speech synthesis) (Demasco & Mineo, 1995). This thesis will focus on the language area in the following subsections, giving an overview into the current research in AAC and NLP-based AAC technology. We will look at the knowledge required by these systems and discuss an example of NLP-based AAC technology called Compansion (Demasco & McCoy, 1992).

---

1. These disabilities are primarily caused by afflictions such as cerebral palsy, ALS, etc.

AAC Language Technology Overview

The majority of the research in the language area of AAC can be grouped in the three categories of representation, organization and processing of language. The representation group research concerns vocabulary design (e.g., icons, word board) (Mineo et. al., 1994; Chang et al., 1993; Magnusson & Briem, 1994; Albacete et al., 1994). The organization of language group includes organizational models (Demasco et al., 1994) and access issues (e.g., scanning, direct access) (Koester & Levine, 1994). The processing group includes research on rate enhancement techniques (e.g., letter prediction, word prediction) (Venkatagiri, 1993; Hamilton, 1994; Demasco et al., 1994; McCoy et al., 1994B; Vanderheyden et al., 1994) and language correction (Suri & McCoy, 1993; Morris et al., 1992).

In early research, little attention was paid to the syntactic, semantic or language issues involved (Kraat, 1990) in developing AAC technologies. However, since the late nineteen-eighties, a subarea in AAC research that concentrates on applying Natural Language Processing techniques to enhance AAC communication devices has emerged. The University of Delaware and the University of Dundee are forerunners in this area of research (McCoy & Demasco, 1995). One of the major problems involved in developing intelligent AAC devices is that suddenly the system designer/user need not only be concerned about the usability issues, placement and access to words, but also needs to provide the information about words which is necessary to do intelligent reasoning.

Some Natural Language Processing (NLP) used in AAC Applications

The use of Artificial Intelligence (AI) and Natural Language Processing (NLP) techniques in the development of AAC systems and devices continues to grow both in research laboratories and, more recently, in commercial products. The use of AI/NLP methods in any application area requires significant language knowledge about words and their various morphological forms, syntactic categories (e.g., NOUN, VERB, CONJUNCTION), thematic roles (e.g., AGENT, THEME, LOCATIVE) (Fillmore, 1977) and control information (McHale & Crowter, 1994). Within AAC, the need to support relatively unconstrained message production (in contrast to structured communication as in a database query language) requires that this knowledge be broad as well as detailed.

Some of the underlying NLP work looks at many of the variables involved for a system to understand ill-formed language (Weischedel & Sondheimer, 1983; Jensen et al., 1983). This research includes developing methods to resolve issues such as, unknown words, word sense disambiguation, resolving referents, etc. (Granger, 1983; Fass & Wilks, 1983). In order to handle well-formed input, a system can generally resolve most of these variables, as long as the domain of the application is specific. In the case of ill-formed input, however, there are additional obstacles to consider (e.g., filling in missing words and/or punctuation, ad hoc abbreviations, lack of tense agreement, causality violation, goal violation, object or event referenced out of context, subject-verb disagreement, homonyms confused). In order to deal with irregularities of input, NLP researchers have been required to develop new methodologies (Small & Rieger, 1982).

Need for Word Knowledge in NLP based AAC Systems

Many word-based AAC systems contain words and word phrases written in traditional orthography. Other systems are based on graphic or icon-based conceptual representations. The communication devices designed for children with disabilities tend to be icon-based; one of their functions is to enable children to understand that language can be used to communicate desires and needs, such as getting help, obtaining objects, expressing likes and dislikes (von Tetzchner, 1988). However, as the child progresses, she/he requires new methods to communicate at her/his increased vocabulary levels. One method to help accommodate this is through spelling, another would be through learning more creative and complicated sequences of icons.

There are two predominate icon-based languages used in AAC today. One is the Blisssymbolic[2] language (Archer, 1977) and the other is the iconic language of the Minspeak system which is based upon the principle of semantic compaction[3] (Baker, 1982). In particular, Minspeak provides an excellent example of a system developed with linguistic and syntactic principles in mind. It will be the focus of the following discussion.

One major focus in the design of the Minspeak language is finding conceptually associated iconic representations and then combining these icons with syntactically bound

---

2. Blissymbolics is a communication system designed by Charles Bliss based on symbols. It was originally thought to be an international symbol system to minimize misunderstandings between people of different language groups (e.g., Chinese, English, and Japanese speakers). Since 1971, Blisssymbolic has been used on communication devices by children with speech disabilities (Magnusson & Briem, 1994).

3. Semantic compaction is the process involving mapping concepts on to multi-meaning icon sequences and using these sequences to retrieve messages stored in a computerized system. It was developed by Bruce Baker.

icons. For instance, a sequence of the APPLE icon and the VERB icon will produce the word *eat*, while a sequence of the APPLE icon and the NOUN icon will produce the word *food*. Minspeak considers the icon images to act as indexes to concepts and these concepts translate to words depending on the conceptual mapping of the iconic sequences (Chang et al., 1992). The researchers involved with developing the iconic languages used by Minspeak have recently been studying the use of AI techniques to introduce semantics to the language design. They have introduced some components from Conceptual Dependency Theory[4] to enhance the development of appropriate iconic sequences for use in programming the device (Chang et al., 1993; Albacete et al., 1994).

Kraat's concerns about the aphasic community (Kraat, 1990) has pointed out that AAC has not addressed the issue of semantic meanings in language; this concern is being addressed by researchers such as Baker and Magnusson who are beginning to incorporate AI techniques into Minspeak and Blisssymbolic. The BlissGrammar program, for example, will correct sentences made with Blissymbols so that the text-to-speech converter will speak or print out easily understood sentences (Magnusson & Briem, 1994). Other AAC researchers have employed AI techniques to word prediction (VanDyke, 1991), letter-based abbreviation expansion (Stum & Demasco, 1992), word-based systems (McCoy & Demasco, 1995), and sentence retrieval techniques (Waller et al., 1992). In order to incorporate the AI/NLP techniques to these methods, a great deal of knowledge is required.

---

4. Conceptual Dependency (CD) is a theory of Natural Language Processing developed by Roger Schank that provides computer programs with the capability of "understanding" natural language sentences by representing sentences in a series of primitives. The goal of CD is to represent language such that it aids drawing inferences from sentences and is independent of the language (e.g., Chinese, English, French) (Rich & Knight, 1991).

Word and letter prediction systems need to be able to access frequency information.[5] Letter-based abbreviation expansion systems require a set of rules and frequency information. One of the more sophisticated NLP-based AAC systems, Compansion (Demasco & McCoy, 1992), was developed at the University of Delaware and the Center for Applied Science and Engineering. Compansion is an example of a word-based system that requires a great deal of knowledge in order to do its processing. That system and its necessary knowledge is described in the following section.

Compansion

One example of an intelligent AAC technique is Compansion, an approach that takes telegraphic input from a user and expands it into a syntactically and semantically well-formed sentence. The Compansion technique assumes a communication system based on words, pictures, or icons (i.e., non-spelling) and attempts to enhance the user's message production rate[6] by requiring only the selection of content words. One advantage of such a system is that it reduces the need to represent morphological information (e.g., verb inflections, endings). This is potentially very beneficial for systems that use picture-based representations.

---

5. Most frequency information is derived from either bigram and trigram frequencies or from corpora information. A study of simulated experiments at the Applied Science and Engineering Laboratories showed that using corpora frequency data, predictions after the first letter were correct 44.8% vs. a correct rating of 27.9% using bigram prediction (Hamilton, 1994).

6. While the Compansion techniques has been primarily described as a rate enhancement technique, it also has potential applications in helping users learn how to produce grammatical sentences.

The Companion system uses both syntactic and semantic knowledge in its processing. The system's processing consists of three major phases. The first phase is the "word order" parser and relies on a syntactic grammar which captures the regularity of the telegraphic input in this initial parsing phase. The word order parser is responsible for grouping words into sentence-sized chunks and indicating each word's part of speech. It is also responsible for attaching modifiers (e.g., compound possessives, adjectives and adverbs) to the word they are most likely modifying. The second phase consists of the "semantic" parser which reasons about the meaning of the content words of the sentence-sized chunks and develops a semantic representation for each chunk. The final phase is the translator/generator which takes the output of the semantic parser and generates an English sentence using a syntactic grammar of English. Here we focus on the knowledge used by the semantic parser since it requires the most sophisticated knowledge sources.

The semantic parser takes a set of words and attempts to fit these items into a well-formed semantic structure, thus determining the intended meaning. In the current implementation, processing is non-incremental; all of the input words are taken together and a semantic representation is created which best accommodates the set of words as a whole. Generally there will be at most one word identified as the main verb in the input words; the parser must determine which semantic role is being played by the other words. Consider the processing of the input *John break window hammer.* Once *break* is identified as the verb, the parser must decide which word of the input represents the agent/experiencer (i.e., person or thing doing the action), which represents the theme (i.e., thing being acted upon), and so on (Fillmore, 1977). This information is represented in the semantic parser in

the form of a case frame that includes preference information (Fass & Wilks, 1983) to help the system reason about the possible intended use for each word of input. For example, a simplified form of the specified case frame for *break* is shown in Figure 2.1 below:

```
verb - break
agexp [toFillPref 4] [[human 3] [animate 2] [ergative 2]]
theme [toFillPref 3] [[physical 3] [fragile 4] [object 1]]
instr [toFillPref 2] [[tool_box 3] [tool 3] [solid 1]]
goal  [toFillPref 1] [human 3]
benef [toFillPref 1] [[human 3] [organization 2] [animate 2]]
loc   [toFillPref 1] [place 4]
timee [toFillPref 1] [time 4]
```

Figure 2.1 Specified Case Frame for the Verb BREAK

The above frame captures two kinds of preferences used by the system. The first kind of preference, the case importance preference, is indicated by the *toFillPref* rating. This indicates which of the roles are most important to fill for a particular verb. In the frame above, the important roles to fill are the AGEXP (agent/experiencer), THEME, and INSTR (instrument). This is indicated by their respective ratings of four, three and two. A four indicates a very high preference while decreasing numbers indicate less preferred, but possible, options.The second kind of preference rating is the case filler preferences which are motivated by Preference Semantics (Wilks, 1975). They indicate what kinds of objects could fill a role and indicate a rating on the "goodness" of each type of filler. The frame in Figure 2.1 indicates that the AGEXP role is preferred to be filled by a human with a rating of three, but that any animate object or ergative object (e.g., a *car*) would also be acceptable with ratings of two. The THEME role is preferred to be filled by a fragile object, but

a physical or abstract object could also serve as a filler, with ratings of four, three, and one respectively. A third kind of preference used by the system, but not directly illustrated in the Figure 2.1, is the higher-order case preferences. This case preference is intended to account for interactions between cases and their fillers. For instance, if a non-human animate (e.g., *dog*) is the AGEXP of a material process (e.g., *eating*), then it is highly unlikely that an INSTR is being used; however if the AGEXP is a human, an INSTR would be very likely. In the semantic parser, this preference is captured by a rule which subtracts from the overall goodness of an interpretation when these conditions are found. An interpretation is given a rating by adding together the numbers obtained by multiplying the case filler and case importance ratings for each word's case (and then subtracting off the number indicated by higher-order case preferences). So for the input *John break window hammer*, we prefer that *John* be the AGEXP (with a rating of 4 times 3), *window* be the THEME (with a rating of 3 times 4) and *hammer* be the INSTR (with a rating of 2 times 3), giving us an overall preference rating of 30. The idiosyncratic case constraints are not used to add or subtract from a case frame's preference rating, but are intended to capture mandatory and forbidden cases within a frame. For instance, a mandatory feature of the verb *hit* is the THEME, while GOAL is a forbidden feature for this same verb. These constraints are placed directly on the verb frames by the absence or presence of these cases (McCoy et al., 1994A).

The basic idea of the semantic parser is to fit the non-verb words of input into the case frame in the best way possible. In order to do this, the semantic parser must access type-information associated with each word. For instance, it must be able to tell that *John*

is a human, that *hammer* is not a human but a physical object, and that a *window* is fragile. With this information the semantic parser can reason about the words of input and generate the sentence *John breaks the window with a hammer.* A major problem for Compansion as a viable AAC system is the amount of information it requires about each word it may encounter. This information is not readily available in any standard database or online dictionary, and it must currently be specified by an NLP researcher. This, of course, greatly hampers the extendability and customization of the system.

Computational Lexical Semantic Overview

Computational Semantics is a cross-disciplinary research area focusing on enabling computers to reason about a language's semantics. This area is comprised of NLP computer scientists, linguists, and psycholinguists and the focus is on developing methods for determining the "meaning" of natural language sentences. Many Computational Semanticists have devoted their time to determining appropriate meanings for component parts (Jackendoff, 1978; Levin & Pinker, 1992; Pustejovsky, 1991) (e.g., the various meanings for prepositional "location" phrases). Because many believe that the meaning of a sentence is primarily influenced by the main verb, much of the work has been devoted to determining the meanings of particular verbs and how they influence sentence meanings (Levin, 1992; Palmer & Polguere, 1994; Levin & Hovav, 1992). Others have studied how language and word/sentence meanings are acquired (Levin & Pinker, 1992; Jubak, 1992). The methodology in Computational Lexical Semantics is to study the acquisition and use of component pieces of language and identify how these pieces influence the meaning of

the whole sentence. Because these component pieces are made up of words, much work has been devoted to the study of appropriate lexical meaning that could explain their meaning. The ultimate goal of lexical semantics is to develop lexicons (containing appropriate meaningful information), methods, and tools that will produce multi-purpose systems capable of handling unrestricted language (McHale & Crowter, 1994).

Current research into the role that phrasal syntactic properties play in the understanding of words has resulted in a unifying focus, leading the community to believe that the core elements of semantic representations are becoming clearer (Levin & Pinker, 1992). This has in turn led to increased attention being focused on lexicon representations (Church, 1994; Macleod et al., 1994), the application of computational and statistical techniques to corpora and machine-readable dictionaries (Hogan & Levin, 1994), and to new tools for the study of lexical representation (Berwick et al., 1994; Zickus, 1994).

NLP systems require knowledge about the words they must process. This knowledge is difficult to acquire and represent in a machine, and has definite effects on system performance, leading to the term "lexical bottleneck" (Byrd, 1989). Current Computational Lexical Semantics (CLS) research aimed at resolving this bottleneck has led to areas of specialization that may or may not need to be applied in conjunction with each other (i.e., the problem is too big to be attacked from one front, so the work has been split up into various areas in the hope that by combining efforts from one or more of these specialized groups, a solution will be found). There are groups seeking to improve the quality of computerized lexicons by basing them on psycholinguistic principles (Miller et al., 1993; Miller & Fellbaum, 1992). Another group is attempting to further refine these improved

lexicons and/or other CLS tools by applying them to each other, such as using Beth Levin's work on diathesis alternations in order to further improve WordNet or vice versa (Berwick et al., 1994; Zickus, 1994; Gu, 1994). Other groups are looking for ways to improve syntactic and/or semantic analysis methods through corpus-based lexical research (Church, 1994; Zernik, 1989). Some combine this corpus-based research with improved lexicons such as WordNet for the purpose of syntactic parsing (Macleod et al., 1994). There are groups focusing on defining and isolating the specific needs of Machine Translation systems (Chen & Huang, 1994), and finally, there are groups researching the lexical needs of integrated systems (Gros et al., 1994; McKevitt & Guo, 1994).

While these separate groups are focusing on their specific research, they also maintain a focus on the end result of their work, resolving the lexical bottleneck. There have been discussions at various international and national workshops[7] and conferences concerning what is needed for lexical processing, what is currently available, and what needs to be done, if it is feasible. There are two notable research efforts that have been incorporated by many Computational Lexical Semantic groups recently: the work by Beth Levin on defining classes of English verbs based on diathesis alternations, and WordNet, an example of an improved lexicon developed by researchers at Princeton University. The work by Beth Levin separates verbs into distinct verb classes based on the syntactic phenomena

7. The International Post-COLING94 Workshop on the Directions of Lexical Research in Beijing, China was devoted to these issues, as is the upcoming 1st Annual Workshop for the IFIP Working Group for Natural Language Processing and Knowledge Representation in April 1995 at the University of Pennsylvania. Other discussions include the SIGLEX Workshop at Berkeley in June 1991 and a workshop on the Representation and Acquisition of Lexical Knowledge: Polysemy, Ambiguity, and Generative at the AAAI 1995 Spring Symposium Series at Stanford University in March 1995.

in which a verb may participate (Levin, 1993). This class differentiation research is on-going and could prove to be an important tool for generating systems that can extract semantic meaning from language and/or generate syntactically and semantically correct language. WordNet is the main lexical resource accessed by LAD and will be described in the next section on dictionaries.

Dictionaries and Lexicons

Dictionaries have been around for a long time and have developed relatively standardized formats through years of publishing (Vizetelly, 1915). Though the shape, size and organizational structure has changed over the centuries, the main role of the dictionary has not. Dictionaries provide users with a way of locating definition, morphological, relational, antonymical, syllabic and phonetic information about words. This section will look at the format and function of paper-based dictionaries, on-line dictionaries, lexicons and WordNet, and conclude with a discussion of other efforts in merging lexical resources.

Paper-Based

While there are various paper-based versions of dictionaries available today (e.g., Websters, American Heritage), they all have similar organizational formats. The words are stored in alphabetical order, and they contain a wide range of information about words such as spelling, pronunciation, inflected and derivative forms, etymology, parts of speech, definitions, illustrative uses of alternate senses, synonyms, antonyms, and special usage examples. This information has proven useful to people over the years. There are, howev-

er, some difficulties in locating certain types of information in paper-based dictionaries. For instance, if you look up the word *poplar*, you will find out that a *poplar* is a kind of *tree*, but in order to locate coordinate[8] terms for *poplar* (an example of a lateral kind of search), you need to start at the beginning of the dictionary and look at the definitions of every word to see if they are also types of *trees*. Thus using paper-based dictionaries, you can search vertically for superordinate terms or subordinate terms without too much difficulty, but to search laterally takes a great deal of effort.

On-Line

With the advent of computer database systems, it was a logical step to create on-line dictionaries. The data can be stored in ways to link dictionary items not only in a hierarchical manner, but also in a lateral manner, thus reducing the time for lexical searches and increasing the flexibility of the searches. It wasn't long, though, before researchers decided to augment the information maintained within on-line lexicons to provide information needed for systems designed to process language. This led to the on-going debate concerning what approach to take in order to store both conceptual meaning[9] and collocative meaning.[10] One school of thought believes that the best way to encode semantic

---

8. Coordinate terms, or sister terms, are terms that are derived from the same parent, or superordinate. They can be thought of as kind-of terms of that parent. (e.g., poplar, ash, maple are all coordinate terms and are kind-of trees).

9. Conceptual meaning is the cognitive content of words. This captures the expression of phenomena that are deeply embedded in the language.

10. Collocative meaning is the communication of meaning via associations between words or word classes. It attempts to explain words in terms of their relations with other lexical items and word classes.

knowledge is through corpora analysis (Velardi et al, 1991). These researchers don't store words as in a normal dictionary, but use corpora to produce conceptually related concepts which they then use to interpret the meaning of words taken in context. Examples of conceptually related concepts (CRC's) are an *activity* is related to a *location* which is a *place,* a *change* is related to a *final state* which is a *product,* and *farming* is related to a *location* which is a *field.* They build semantic knowledge bases of these CRC's, in place of lexicons, from corpora analysis. The resulting semantic knowledge base can then be used in semantic processing for domain-specific applications. The other school of thought is to design on-line lexicons that incorporate additional knowledge necessary for deriving conceptual and collocative meanings (Miller et al., 1993). There are two on-line lexicons commonly being used by the research community, these are the Longman's Dictionary of Contemporary English (LDOCE) and WordNet. The next section describes WordNet in more detail.

WordNet

WordNet (Miller et al., 1993) is an on-line dictionary/thesaurus developed by a group of psycholinguists at Princeton University with over 95,000 lexical entries, 51,000 being simple words and 44,000 being collocations (e.g., *attorney general*). WordNet has the ability to distinguish a number of different senses of words and to produce synonym sets, called *synsets,*[11] and sentence glosses for these differing senses. The primary purpose

11. A *synset* is a grouping of synonymous words (i.e., words with the same meaning). An example of a *synset* is [merchandise, wares, product].

of a dictionary is to provide definitions for words. WordNet provides definitions for all synsets and thus for every sense of every word.

WordNet incorporates pyscholinguistic theories and research attempting to mimic how people organize lexical memory (Miller et al., 1993) in its design. The central organizing feature for the lexical entries is a lexical matrix: a mapping between words and the synsets to which they belong. There are four different word databases associated with WordNet: one for nouns, one for verbs, one for adverbs and one for adjectives; each one having a slightly different hierarchical representation. This hierarchical organizational method is very natural for nouns,[12] as it is based on psycholinguistic findings that people store nouns in their memories in a hierarchical manner from specific to general (Miller, 1993). WordNet has isolated 25 unique beginner synsets (see Figure 2.2), which are used to build up the hierarchical structures for nouns (see Figure 2.3).

```
[act, action, activity]      [natural object]
[animal, fauna]              [natural phenomenon]
[artifact]                   [person, human being]
[attribute, property]        [plant, flora]
[body, corpus]               [possession]
[cognition, knowledge]       [process]
[communication]              [quantity, amount]
[event, happening]           [relation]
[feeling, emotion]           [shape]
[food]                       [state, condition]
[group, collection]          [substance]
[location, place]            [time]
[motive]
```

Figure 2.2 List of 25 Unique Beginners for WordNet Nouns

12. Each sense of a noun has at most one superordinate term, thus mapping easily onto a tree-like hierarchical structure.

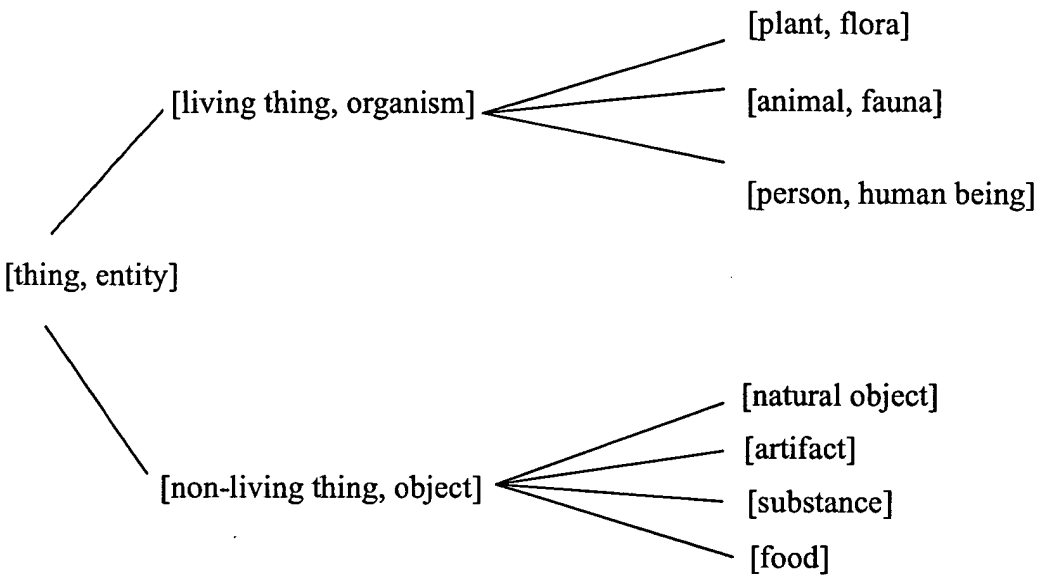


Figure 2.3 Hyponymic Relations of Seven WordNet Unique Beginners

WordNet uses a lexical inheritance system in its hierarchical structuring of nouns. This allows for semantic components to be inherited from their superordinate/parent terms. The base synset for most hierarchies is [thing, entity]. As mentioned previously, WordNet stores its words in synsets. In so doing, a noun is stored in as many synsets as there are different senses to the word. For instance, *bank* is in 6 separate synsets, (e.g., Sense 1, 2, 3 - [bank], Sense 4 - [depository financial institution, bank], Sense 5 - [bank, supply, reserve], and Sense 6 - [bank, bank building]). Notice how the first three synsets look identical on this level. However, if you look at their superordinate terms, the differing senses of the word become evident. The superordinate term synset for Sense 1 of *bank* is [slope, incline], for Sense 2 is [ridge] and for Sense 3 is [array, arrangement]. By placing

nouns into synsets in this manner, WordNet can be used by systems to enhance word sense disambiguation. One of WordNet's strengths is the fact that it has all of its over 95,000 words broken into their different senses. However, this strength is also a weakness in that there is no agreed upon standardization among the lexical research community as to the number of senses any given word has, or what label should be given to these senses (Ide & Veronis, 1994). As an example, Martha Palmer is a researcher who has spent years defining the different senses of the word *break* (Palmer, 1990; Palmer & Polguere, 1994). Her work has merged the 30 different paper-based dictionary meanings into four core senses for *break*. WordNet, on the other hand, has 23 different senses for *break*, although some of the discrepancies between WordNet's different senses are difficult to distinguish (e.g., Sense 1 - violate, fail to agree with, go against, break, be in violation of; Sense 2 - transgress, violate, go against, breach, break, be in violation of). It is important though, to have some way to disambiguate word meanings in order to facilitate semantic reasoning. For instance, if we are talking about the word *bank*, are we talking about the financial institution where we make monetary transactions or are we talking about the bank by the river where we go fishing or swimming or picnicking? To this end, the sense information in WordNet is a valuable lexical tool.

As mentioned previously, there are some limitations with traditional dictionaries. One such shortcoming is that the information stored with a word is often incomplete. When one looks up a noun, for example *platypus*, one learns that it is a semiaquatic, egg-laying mammal, but unless one is an expert on mammals, there is no way, other than by looking up mammal, to find out if a *platypus* has hair. Dictionaries are ordered alphabeti-

cally and not grouped semantically, therefore such searches can be cumbersome. This weakness in contemporary dictionaries demonstrates one of the major strengths of WordNet: its semantic and lexical relations. By using the WordNet on-line lexicon, it is easy to discover the attributes of a given noun by traversing the semantic links of its superordinate term (i.e., its parent).

WordNet's semantic and lexical links also enable searching for various relationships to other words in the English language. One of these relationships is the **is-a** hierarchical relationship, which is one of the most pivotal in giving meaning to words that are related to each other. For instance if we know what a *vehicle* is, then it helps us to understand what a *car* is by knowing that a *car* **is-a** *vehicle*. This information is useful for applications that need semantic relationships between words, such as systems designed to teach English as a second language.

Another important lexically-linked feature needed by many lexical systems is a means by which you can discover what parts or attributes a word possesses. This is often referred to as the **has-a** relationship. As in the case of the **is-a** relationship, the WordNet database has semantic links to help derive this information. For instance, using **has-a** knowledge you can determine that a *car* has an accelerator, gun, throttle, automobile engine, boot, luggage compartment, trunk, fender, bumper, car door, car mirror, car window, mudguard, first gear, low gear, floorboard, glove compartment, gear box, transmission, radiator grill and so on, all using WordNet's semantic and lexical links.

Another feature lacking in contemporary dictionaries, but available through WordNet, is information about coordinate terms (i.e., sister terms). Someone looking for infor-

mation about other mammals would be forced to search the dictionary from beginning to end looking for terms that are classified as mammals. The prototypical lexical entry for a word points to its superordinate term, not laterally to its coordinate terms or downwards to its hyponyms (i.e., its children or subordinate terms). This is one of the strengths of WordNet: its ability to reach related terms easily through its direct links to superordinate, coordinate and hyponymic terms makes searches of such information routine.

The familiarity rating of a word is a useful indication for the probability of use for a word. In order to incorporate familiarity into WordNet, the designers have attached a syntactically tagged index of familiarity to each word. Generally, frequency is used as an indication of familiarity for words, however, WordNet's designers note that the relationship between frequency of occurrence and polysemy has been well-documented ever since 1945 (Miller, 1993). They go on to contend that polysemy predicts the amount of lexical access as well as frequency does, since the more frequently a word is used, the more different meanings it has in dictionaries, therefore, WordNet uses polysemy as its familiarity index. The familiarity rating is a useful way for users to locate a suitable alternative choice for a word simply by traversing a word's hierarchical links and finding the word with the highest familiarity rating. For instance, by traversing the WordNet hierarchical links for the word *diary* which has a familiarity count of 2, you can find the alternative choice of *writing* or *writings* which has a familiarity count of 7. This is a useful feature for systems interested in writing, generation, or teaching English as a second language. This familiarity feature is not only available for nouns in WordNet, but also for the verb hierarchy.

Due to the nature of verbs, the complexity of their predicate argument structures (e.g., noun phrases), and other verb features (e.g., entailment, polysemy), it is unreasonable to represent verb entries in the lexicon merely with synsets (Miller & Fellbaum, 1992). To compensate for the complex nature of verbs, WordNet organizes them into senses based both on *synsets* and other verb features. It also provides short glosses to indicate the "meaning" of each sense (Fellbaum, 1993). WordNet was not designed to recognize the syntactic regularities that are a part of the semantic meaning of verbs; to compensate for this, WordNet includes one or more generic verb sentence frames for each verb *synset*. These frames distinguish features of the verbs by demonstrating the kinds of sentences in which they may participate. These frames indicate specific features of verbs that have been highlighted by researchers such as Beth Levin (e.g., argument structure, prepositional phrases and adjuncts, sentential complements and animacy of the noun arguments) (Miller & Fellbaum, 1992). The sentence frames are limited to a standard, generic format that is shown in Figure 2.4 (for a complete list of the WordNet verb frames, see Appendix D).

Somebody ----s something
Something ----s
Something is ----ing PP
Something ----s something
Something ----s somebody
Somebody ----s somebody
Somebody ----s
Somebody ----s something to somebody
Somebody ----s that CLAUSE

Figure 2.4 WordNet Verb Sentence Frames

WordNet is an exceptional on-line dictionary, containing a great depth and breadth of lexical and semantic knowledge. It is currently being used by many researchers

(Church, 1994; Macleod et al., 1994; Berwick et al., 1994; Zickus, 1994; Sutcliffe et al., 1994) as a tool to help solve some of the basic road-blocks in Computational Lexical Semantics and NLP, especially as a means of word sense disambiguation.

WordNet does have some weaknesses. The morphology information within WordNet is minimal and it only goes in one direction: stripping off endings or searching exception lists to find the root form of a word (e.g., *went* is changed to *go* during a WordNet database search). There is no morphological mechanism for adding endings to root words (e.g., being able to pluralize *story* to *stories*). Morphological information is important to systems involved in Natural Language Processing/Generation. Additionally, if someone desires phonetic information (e.g., speech synthesis systems), information on non-noun/verb/adjective/adverb terms (e.g., word-based systems), proper nouns, or information on function words; they need to go to another source.

WordNet makes a nice base for developing a multi-purpose linguistic tool. It has the traditional dictionary information (e.g., definitions, example sentences, part of speech tags, illustrative uses of alternate senses, synonyms, antonyms), and it takes advantage of computer technology by incorporating semantically related links within its lexicon to provide additional lexical information. WordNet is the primary dictionary resource accessed by the Language Access Database (LAD) which will be described in Chapter 3.

Efforts in Merging Lexical Resources

There are other current research efforts in merging lexical resources besides the LAD project. While the two discussed here have similarities with LAD, there are signifi-

cant differences in approach, design and functionality. The first work is being developed by Kevin Knight and Steve Luk at USC/Information Sciences Institute and the second work is by Michael McHale and John Crowter at Griffiss Air Force Base in New York. Since both works use the Longman's Dictionary of Contemporary English (LDOCE), a brief description of it is provided before further discussion proceeds.

LDOCE is designed to be a learner's dictionary of English as a second language. It contains over 27,000 words, chosen for their core vocabulary aspects and frequency of current usage, and over 70,000 word senses. It includes short definitions, examples of usage, syntactic categories (e.g., adj followed by *to*), semantic categories (e.g., human, animate object), and pragmatic categories (e.g., economics and business). One of the nice features of LDOCE, from the angle of incorporating it into other computerized systems, is that it has a controlled defining vocabulary of approximately 2000 basic words. This allows systems to need less word knowledge while processing the definitions; however, this can also lead to more complex syntactic representational structures in order to fully define words with the limitations of a constrained vocabulary.

Knight and Luk (1994) are building a large-scale knowledge base for machine translation using semi-automatic methods for manipulating and merging existing resources. Their resources are: 1) the PENMAN Upper Model which is a network of about 200 nodes to influence linguistic choices; 2) the ONTOS model from Carnegie Mellon University which is an ontology designed to support machine translation; 3) LDOCE; 4) WordNet; and 5) the Harper-Collins Spanish-English bilingual dictionary containing thousands of Spanish words with English translations. Their methodology is to merge these on-line

dictionaries, semantic networks and bilingual resources through semi-automatic methods (e.g., conceptual matching of semantic taxonomies, definitions). By using the resources chosen, they have access to much of the semantic and syntactic information needed by NLP-based systems. These include semantic categories (e.g., human, inanimate object, tool), sense information, and phonetic information (though they do not mention it, LDOCE provides pronunciation data on its words). This work has not been completed. Most of their work so far has been in developing the merging algorithms needed in order to match up the different resources. No complete testing has been done, therefore evaluation of this system is not possible.

The goal of the second research group is to construct a lexicon from a machine readable dictionary. The resources used are: 1) a Principle-Based Parser (PBP), based on Chomsky's Government-Binding Theory; 2) LDOCE; and 3) Roget's International Thesaurus, with approximately 225,000 words. A PBP is based on a lexical theory of grammar and needs more information than most parsers require (e.g., morphological forms, part of speech, syntactic complements, thematic roles and control information). The combination of LDOCE and Roget's seems to provide the information necessary for the PBP. One of the limitations of this system is with regard to thematic roles (e.g., AGENT, THEME, BENEFACTIVE, EXPERIENTIAL, and LOCATIVE). LDOCE does not explicitly provide thematic role information. McHale and Crowter (1994) used a method of searching for repeated patterns of words in the word definitions in LDOCE in order to extract this information (e.g., the pattern *to cause to* is indicative of an AGENT role). This only proved successful for two-third's of the verbs; of these, ten percent had conflicting patterns and

had to be re-analyzed by hand. They were also limited to the five thematic roles listed above. They developed a mapping facility that correctly mapped sixty-three percent of the word senses in Roget's Thesaurus to the word definitions in LDOCE. This work is not completed. However they did implement a lexical browser (limited to aerospace terminology) that will allow a user to select a word which produces the word's definition and links to any aerospace terms. The browser also has speech output in the form of uttering the word and a sample sentence from LDOCE.

The system being designed by Knight and Luk (1994) is for the specific domain of machine translation and has not addressed the needs of other domains. Their design of merging the five specified resources together limits their system's extensibility. In other words, if they wish to add a feature, such as frequency, they need to re-work a great deal of their system in order to merge in additional resources. They have addressed the domain-specific needs of machine translation systems, but have not provided for the needs of other NLP-based systems. While McHale and Crowter (1994) were aiming at providing NLP technology with the ability to handle unconstrained language, they had to cut back on these aims due to personnel and time restrictions and thus have fallen short of their original goals. Their system is designed to provide a large, general lexicon for a broad coverage, domain-independent, syntactic parser (PBP); however, it does not address the needs of some of the NLP-based systems discussed earlier. What is needed is a system with a wider scope of information, a more extensible architecture (for incorporation of future lexical and linguistic resources), that contains semantic and syntactic information capable of supplying the needs of several real-world applications.

Chapter 3

LAD DESIGN

Motivation

One of the limitations of AAC devices today is the size and lack of information available in their dictionaries. While some may contain an adequate amount of words, none of them contain sufficient information on these words to do semantic and syntactic reasoning, such as the word information needed for a semantic parser. In addition, while there is substantial interest in the development of natural language interfaces within the general software community, there currently do not exist any lexical databases that provide both a broad coverage (in terms of numbers of words) and sufficient depth of information (e.g., case frames) for individual words (Zickus et al., 1995; McHale & Crowter, 1994).

There are several lexical tools and systems available on the market today; however, there are limitations with most, if not all, of them. They each have their own strengths, weaknesses, and specializations. In order to take advantage of more than one tool at a time, there needs to be a centralized interface system that will extract and glean the desired information in a consistent, understandable and functional manner. Thus the idea behind the Language Access Database (LAD) was developed.

The approach to designing LAD has been to create an implementation with C++ and Lisp[1] interfaces that allows a programmer to access several different databases (or lexical resources) in a seamless manner. LAD provides the programmer with a set of functions which can be used to retrieve various types of information. LAD then queries appropriate databases, retrieves the desired information, and returns it to the user. Thus the user is given the impression that a single database containing a variety of information is available, while LAD may actually access several different lexical resources to handle a query.

At the same time, the programmer is given as much or as little control as they need. For instance, a user can simply query LAD about the frequency of a word and LAD will return the frequency rating found for the most generally accepted meaning of that word in some default corpus. Alternatively, if the programmer prefers, LAD provides mechanisms for the user to specify a specific "sense" of the word they are interested in and/or specify which corpora they would like to use. Thus it provides ways for users to override its default settings and to exhibit a great deal of control over the way a specific query is processed.

LAD accesses several different lexical resources, the most unusual of these being the on-line dictionary/thesaurus WordNet, that was discussed in the previous chapter. It is WordNet that contains much of the semantic information needed for intelligent AAC applications. This chapter will describe the functionality of the LAD design, the lexical re-

1. In our NLP laboratories, we often use Lisp to develop prototypes and C++ for commercial application development.

sources available to LAD, the LAD mapping functionality for different application areas, the object-oriented design of LAD, its extensibility, and finally its current implementation status.

Functional Interface

LAD is designed to be a lexical resource for a variety of applications. A goal of this work is to provide a functional interface that allows a user to query for several different types of lexical information. Often the methods provided allow for optional arguments that give the programmer more control over the way the query is evaluated. In this section, the LAD design functionality will be described. This section has been broken into four sub-sections; the first one describes the argument structure of the LAD methods and the next three reflect the "kind" of lexical knowledge being returned (e.g., semantic, syntactic, and other specialized knowledge) by the LAD functions (for more detailed LAD specifications, see Appendix A).

Argument Structure

LAD was designed based on the object-oriented paradigm. This gives the system the capability of having a flexible argument structure, providing LAD with an adaptability and robustness that otherwise would have been difficult to obtain. All of the functions described in the next three sub-sections have the over-loaded argument capability of being able to add more detailed query specifications. This means that every function, besides being able to provide the more generic information requested for a word, can also return in-

formation for a specific sense of a word, part of speech for a word, and lexical source used for the information retrieval. For instance, an **is-a(book, dramatic_composition)** would search all senses of the word *book* to see if *dramatic_composition* is in its hierarchical lexical links, while **is-a(book, dramatic_composition, 2)** would only search (WordNet) sense 2 of *book* for this superordinate. Frequency information searches might add either part of speech and/or source information into its query, such as **frequency(and, CONJUNCTION)** or **frequency(writes,VERB, mobywords)**. This concept of providing specific information for queries is an important one for the LAD system. It enables the capacity to return both generic and specific information on words from its lexical resources. The next three sections describe the functions of LAD; the kind of argument(s) they take and the output they yield is generally illustrated with examples.

Functions for Semantic Knowledge

is-a(word, parent)

This function takes two arguments; the word that **is-a** information is needed for and the parent-term being considered for the **is-a** query. For instance if we know what a *vehicle* is, then it helps us to understand what a *car* is by knowing that a *car* **is a** *vehicle*. The output from this function is an integer, 0 for false and 1 for true. In the example shown, the output is a 1 for true. If the query was for the input *book, dramatic_composition*, and 2 (indicating sense 2 of *book*), the answer would be 0 for false, since *dramatic_composition* is not part of the hierarchy for *book* as in "record, record-book, book."

is-hierarchy(word)

This function takes one argument, **word**; it returns the **is-a** information associated with the word. For instance, you might know that a *poplar* **is a** *tree*, but you might want to know what other superordinate terms it has. In this case, the output is a character string containing all of the hierarchical links for *poplar* (e.g., "poplar, poplar tree => angiospermous tree, flowering tree => tree => woody plant, ligneous plant => vascular plant, tracheophyte => plant, flora, plant life => life form, organism, being, living thing => entity"). This tells the user system what other categories *poplar* can be considered as an **is-a** term. If multiple hierarchies exist, they are separated, and if sense is specified, only the hierarchy for that sense of the word is returned.

list-semantic-categories(word)

This function takes one argument, the word for which semantic category information is needed. Semantic category information is related to the **is-a** information described in the previous example. In this case, however, the categories returned are limited to those used by the Compansion system (listed in Appendix C). For instance, if we ask for the list of semantic categories *hammer* belongs to, it returns the list, "object, inanimate, tool." The output from this query is a character string.

has-a(word, attribute)

This function takes two arguments, the word that **has-a** information is needed for and the attribute being searched to determine if the word contains it. For instance, if we

want to know if a *car* **has-a** *windshield,* we will use this query. The output from this function is a integer, 0 for false and 1 for true. In the example shown, the output is a 1 for true.

has-parts(word)

This function takes one argument, the word that we want **has-parts** information about. For instance, you might want to know all of the attributes a *car* has. The output from this function is a character string. In this case, the output is a character string containing "car - windshield, engine, trunk, wheels, steering wheel, mirror, fender, accelerator, bumper, floorboard, glove compartment, roof, suspension, tail pipe, turn signal, hood, radiator grille, gears,..." The output for sense 3 of *car* (as in *railway car*) would return "suspension, suspension system."

semantic-properties(word, property)

This function takes two arguments, a word and a semantic property. The goal of this function is provide semantic information about a word; for instance, given the noun *tree*, what is its means of reproduction? In this case the word would be *tree,* the property would be *reproduction*, and the output would be the character string "seeds." Other semantic queries that could be made of the noun *tree* would be *utility* yielding "shade, protection from the wind, produces oxygen, provides fuel, provides wood for construction," or *height* yielding "trees are generally tall." The output of this query is a list of character strings.

definitions(word)

This function takes one argument, the word for which a definition is required. For instance, if the argument is *tree*, the output will be "Sense 1=>tree, tree diagram - a figure that branches from a single root; 'genealogical tree,' Sense 2=>tree - a tall perennial

woody plant having a main trunk and branches forming a distinct elevated crown; includes both gymnosperms and angiosperms." If the function is given the arguments *platypus* and 1 (requesting sense specific information), the output would be "a platypus is a semiaquatic, egg-laying mammal."

familiarity(word)

This function takes one argument, the word needing familiarity information. There is a great deal of research into the polysemous[2] nature of words. Some researchers contend that polysemy is an indication of the frequency of a words usage, while others contend that it has to do with the familiarity of a word. The LAD familiarity rating is based on polysemy. As we have defined familiarity, it indicates a prediction on lexical access. The output is in the form of an integer ranging from 1 (not familiar) to higher ratings, such as 48 (very familiar). The familiarity rating for the word *break* is 45.

sense-info(word)

This function has one argument, the word for which sense information is needed. The output is the number of senses found for that word and is an integer. For example, the output for the sense-info query for the word *break* is 23. This argument does not have a sense specific overloaded function, as it would not be logical to ask how many senses sense 1 of a word has.

alternate-word(word)

---

2. In general, polysemy refers to the multiple meanings some words have, especially verbs.

This function takes as an argument a word for which an alternate choice is needed. Often times when someone is writing a paper, letter or thesis, it is useful to get a good alternative for a word (e.g., when writing about *broncos*, another acceptable term would be *horse*, its generic type). LAD uses the index of familiarity to provide useful alternative words. The output of this query is a character string. For example, given the input *bronco*, LAD will return "horse."

coordinate-terms(word)

This function will take one argument, the word looking for its coordinate (or sister) terms. The output is in the form of a character string. So, if it is given the argument *watch*, it will return "clock, hourglass, sandglass, sundial, timer, atomic clock, ticker." If it is given a specific sense of the word, as in *ash* and the sense is 2 (for a type of wood or tree), the output will include, "yellowwood, balsa wood, boxwood, acacia, redwood, bamboo, poplar, birch..."

is-coordinate-terms(word, sister)

This function will take two arguments, the word being queried about and a possible sister term. The output is an integer, 0 representing false and 1 representing true, indicating if the sister argument is indeed a coordinate term of the word or not. For example, if the first argument is *ash* and the second argument is *poplar*, the result would be 1, or true.

synonyms(word)

This function takes one argument, the word for which synonyms are needed. The output of this function is a character string. For instance if the argument is the word *shoe*, the output would be "footwear, footgear, horseshoe, U-shaped plate, brake shoe." If there

is a sense specifier argument to this function, for example, *break* and 4, the output will be "disperse, dissipate, scatter, break, spread out, break up, come apart, separate, part, split, move apart."

parent-terms(word)

There is one argument to this function call, the word requiring parent-term information. As in previous examples, if a sense specifier argument is provided, then the parent-term for that specified sense of the word will be returned, else it will return all of the parent-terms for all senses of the word. The output from this function is a character string. For instance, if there is one argument, *shoe*, the output will be "{footwear, footgear}, {plate, scale, shell}, {restraint}" and if the arguments are *platypus* and 1, the output will be "mammal."

children-terms(word)

There is one argument for this function call, the word looking for its children-terms. By adding a sense specifier argument, it searches for the specified sense of the word, otherwise it searches for all senses of the word. For instance, if the argument is *lake* with no sense specified, the output will be "{reservoir, artificial lake}, bayou, Great Lakes, Lake Erie, Lake Huron, Lake Ontario, Lake Michigan, Lake Superior, {lagoon, laguna, lagune}, {pond, pool}, Caspian Sea)." The results from this search are character strings. In the case of multiple strings for different senses, they will be returned in a list format to provide a separation between the different strings.

antonyms(word)

There is one argument for this function, the word for which antonym data is needed. All antonyms for the word will be returned, unless there is a sense specifier argument, then the function will return sense specific information. For example, if one argument, *male*, is given, the output will be "woman, female." The output for this function is a character string. In the case of multiple antonyms for different senses, they will be returned in a list format to provide a separation between differing antonym strings.

nominal-relationship(word)

There is one argument to this function, the word for which nominal relationship information is needed. The output of these searches is either a character string or a list of character strings, depending on whether sense specific information is requested. For instance, if the word is *Amazon* and sense 1 is specified, the output will yield "river." If there are multiple answers, as in the case of no sense specification, the answers are separated by being placed in a list.

reverse-nominal-relationship(word)

Alternatively, LAD also has been designed to include a reverse nominal relationship function. This will take one argument, the familiar category name. It will return a character string for its output. For example, if the argument is *satellite*, the output string will be "moon, astronomy satellite, communications satellite, space station, Salyut, Skylab, sputnik, spy satellite, weather satellite, meteorological satellite."

case-frames(verb)

This function has one argument, the verb for which a specified case frame is desired. The output is a **verbFrame**, specified verb case frame object. The particular case

frame contains the information needed for the SRM project described in the next chapter. There are twenty distinct verb case frames (see Appendix B), each having different ratings and semantic category lists. For instance, given the verb *break*, the output would look like:

```
verb - break
agexp   [toFillPref 4] [[human 3][animate 2][ergative 2]]
theme   [toFillPref 3] [[physical 3][fragile 4][object 1]]
instr   [toFillPref 2] [[tool_box 3] [tool 3] [solid 1]]
goal    [toFillPref 1] [human 3]
benef   [toFillPref 1] [[human 3][organization 2][animate 2]]
loc     [toFillPref 1] [place 4]
timee   [toFillPref 1] [time 4]
```

Functions for Syntactic Knowledge

is-part-of-speech(word)

This function takes two arguments, the word that is being searched and a part of speech tag (an overloaded argument). The output is an integer, 0 for false and 1 for true. For instance, if the arguments are *bow* and NOUN, the output will be 1. Similarly, if you have *bow* and VERB, the output would also be 1, for true. However, if the arguments were *bow*, VERB, 9, the output would be 0, since *bow* has 10 senses as a NOUN, but only 4 as a VERB.

get-part-of-speech(word)

This function takes one argument, the word for which you are searching for part of speech information. The output is a string indicating the parts of speech for the word. For instance, if the argument is *bow*, the output would be "NOUN, VERB, ADJECTIVE."

word-compounds(word)

This function takes one argument, the word for which you are searching for compound usages. The output is a character string. For instance, if the argument is *dandelion*, the output would be "common dandelion, dandelion, dandelion green, dwarf dandelion, fall dandelion, krigia dandelion, russian dandelion."

morphology(word, tense, kind)

This function can take up to three arguments. The first argument is the word for which morphological information is desired. The next two arguments specify the kind of morphological information wanted for that word. The output of this function will be a character string. For instance if the arguments are *be, present, first-person-singular*, the output would be "am." The third argument is optional, since not all morphological searches require a third argument, for example, *candy, plural* would yield "candies."

Functions for Other Specialized Knowledge

pronunciation(word)

This function has one argument, the word for which phonetic information is requested. For instance, if the arguments are *lead*, NOUN, sense 9, the output will be the phonetic string for the word as used in the sentence *I need more lead for my pencil.* Whereas, if the arguments are *lead* and VERB, the output will be the phonetic string for *lead* as in *I will lead the girl scout troop tomorrow.*

syllabification(word)

There is one argument for this function. It is the word requiring syllabification information. The output will be a character string. For instance if the argument is *category*, the output will be "cat-e-gor-y."

frequency(word)

This function will take one argument, the word requiring frequency information. The output will be a floating point integer. For instance, the output for *and*, and CONJUNCTION would be 1.68549, or if the arguments were *writes*, VERB, and Moby Words, the output would be 0.211624.

LAD Resources

The LAD project aims at providing a great deal of functional capabilities. While many lexical resources are available today, not one of them provides all of the information LAD requires. Therefore, in order for LAD to accommodate the functionality just discussed, it needs to have access to several linguistic and lexical database resources. The primary LAD resource is the WordNet database. Some of the other sources that LAD can access include an internally developed verb case frame database (where verb frames such as the one from our previous example are stored), a morphology database, a file containing phonetic information, a syllabification file, and databases containing various information derived from the Brown corpus and the Carterette corpus. The following sections cover some of the lexical resources of LAD. It should be noted that LAD has been designed for

extensibility and because of this, augmenting it with additional databases is a fairly straightforward process.

Figure 3.1 shows the overall structure of LAD and the resources it accesses. One important function of LAD is the integration of multiple lexical resources and/or files. These resources are shown on the right part of the figure. The architecture is extensible in that new lexical resources can be added without modification of the database engine.

WordNet

LAD derives a majority of its semantic knowledge from WordNet. The is-a and has-a relationships, word definitions, familiarity, sense-information, alternate-term-information, coordinate terms, synonyms, parent-terms, children-terms, antonyms, nominal-relationships, reverse-nominal relationships, word-compounds and most of the semantic-property information, are all derived from the WordNet lexicon. The hierarchical structure of the WordNet data, as described in Chapter 2, is one of the major reasons so much semantic knowledge can be extracted. This aspect of WordNet combined with its broad lexical coverage (over 95,000 lexical entries), has provided LAD with a vast amount of available knowledge and has proven to be an excellent choice as its primary lexical source.

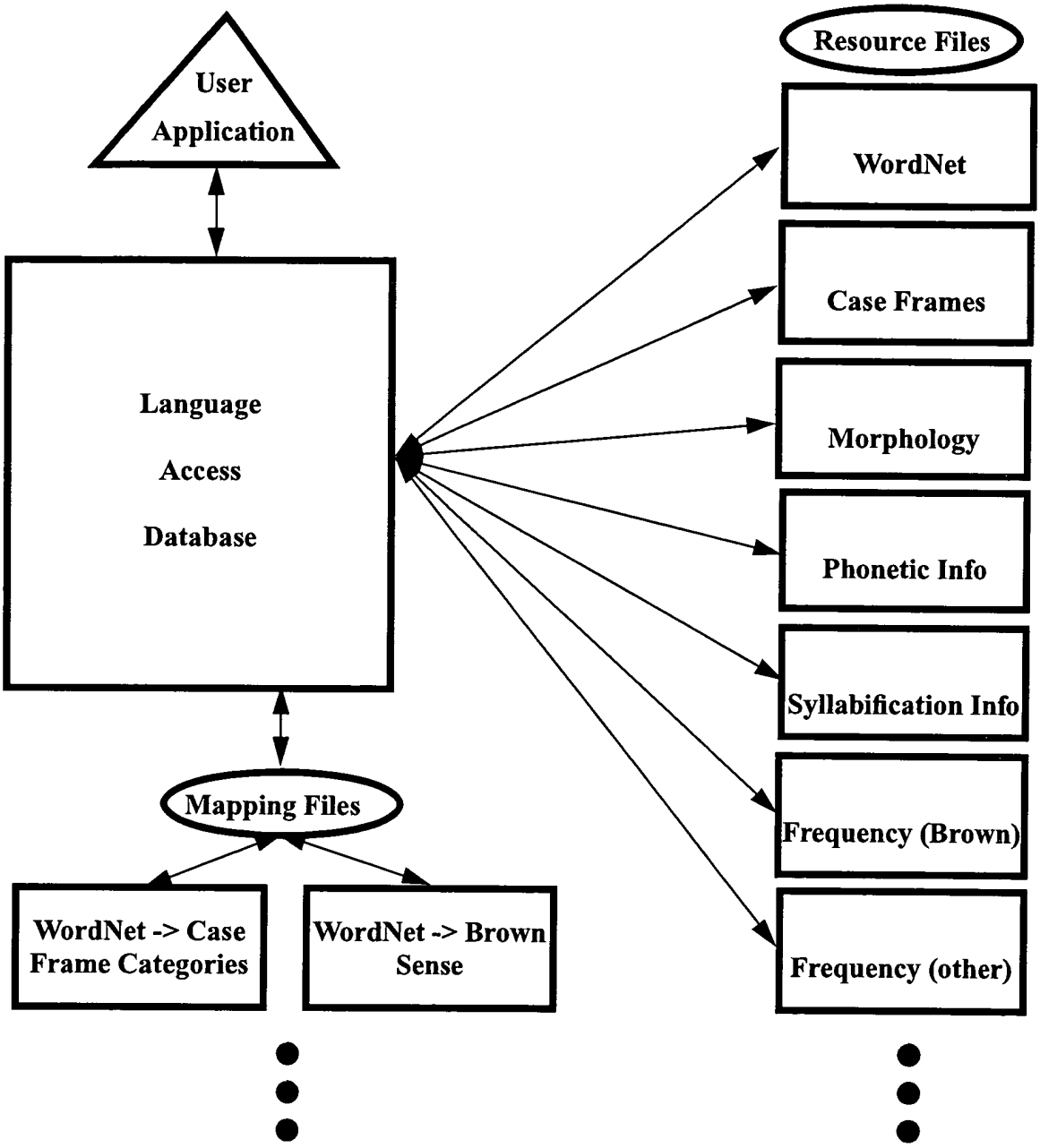


Figure 3.1 Language Access Database Diagram

Case Frames

The case frame information is contained in a secondary database consisting of a flat-file with over 85 verbs and their attached case frames. The case frames contain data used for the semantic reasoning done in the Semantic Reasoning Module (SRM) described in the next chapter. The SRM application was developed to test the LAD design, including the extensibility required for adding additional resources. In addition, the case frame database illustrates how multiple lexical resources can be used to retrieve desired information. For instance, the number of verb case frames that LAD has access to is larger than 85 due to the synonym sets in WordNet. For example, some systems may need verb frame information on verbs that do not have frames (e.g., *pummel*). By default, LAD currently searches for a verb frame from the case frame database. In the case where the verb is not represented in the secondary database, LAD searches for synonyms of the verb from WordNet (e.g., *crush*) and then checks in the case frame database for one of the synonyms.

Morphology

An important aspect of computerized language generation and understanding is the ability to do morphological reasoning about words. For example, if the word of input is *trees*, a system will need the ability to reduce it to the root word *tree* in order to search for information associated with the word in various databases.[3] Likewise, for the generation of language, if the input is *John be happy* or *They be happy*, morphological information is

3. It is normally the case that word information is associated with root words in databases. There are instances, however, such as WordNet which has a morphological processor that reduces words to their root form before it searches the database.

needed to find the third person present singular or third person present plural form of the word *be* in order to generate either *John is happy* or *They are happy.* As you can see from these examples, morphology is an important lexical tool. Much of the needed morphological information is contained in collections of databases provided by Moby Words. However, one might imagine a database which contains only exception data and a processor which handles other morphological information in a rule-based system.[4] Incorporating such a system would be straightforward given the LAD design.

Phonetic Information

The phonetic information LAD will access is contained in the Moby Pronunciator II file from the Moby Word II collection. It contains over 175,000 phonetic entries using the standard International Phonetic Alphabet. Moby Pronunciator II contains many common names and phrases borrowed form other languages, as well as a large number of common English words, compound words and phrases. This resource will provide phonetic information for applications interested in producing speech synthesis or phonetic information.

Syllabification Information

The syllabic information LAD will access is contained in the Moby Hyphenator II file from the Moby Word II collection. It contains 187,175 hyphenated single and com-

4. There is presently a rule-based morphology program with exception tables presently being used in a different application at the Natural Language Processing Laboratory.

pound words. This resource will provide syllabic information for applications requiring hyphenated word information.

Frequency Information

Corpora have been used by various NLP applications in place of large lexicons to enable semantic and contextual word meaning (Velardi et al., 1991). The Brown Corpus is one of the more prominently analyzed and used corporas. It was compiled by W. Nelson Francis and Henry Kurcera at Brown University using written American English sources sampled in 1961. It contains 1,014,294 words from sources that include the press, religion, popular lore, biographies, essays, general fiction, science fiction, adventure fiction, romance stories and humor. It comes in both a syntactically tagged version and an untagged version. The frequency information database will access frequency information from the Brown Corpus, the Carterette Corpus and Moby Word II, which includes a frequency file of the 1,000 most frequently used English words, a file with the 1,000 most commonly used English words on the Internet computer network in 1992, and a file with the 1,185 most frequently occurring substrings in the King James Version Bible.

LAD Mapping

LAD is intended to be a useful semantic and lexical tool for multiple systems, which are designed independently of LAD and LAD's resources. It is anticipated that queries may require some mapping between various sources in order to be fulfilled. For instance a query may require LAD to access one database in order to get information needed

to access another database to properly retrieve the necessary information. In addition, there may be queries which require retrieving similar information from two different databases, but the information itself may be categorized differently in the two databases. In order to handle these naming discrepancies and complex searches, LAD has a mapping capability which allows information between different databases to be used consistently from one to another, even though the information may be given different names in the different databases. The mapping and availability of multiple database accesses are done in a transparent manner to the functional interface.

Semantic Categories

A semantic category is a way of expressing attributes of a word or the roles a word can fill (e.g,. song -> auditory, auditory communication, auditory sensation). The Semantic Reason Module (SRM), described in Chapter 4, requires a list of semantic categories a word can fill in order to semantically reason about possible word roles. LAD facilitates the SRM by searching the WordNet database to determine what categories can be attributed to a word. However, there are some semantic categories in the SRM that have different names in the WordNet database (e.g., SRM - ingestible, WordNet - foodstuff, nutrient; SRM - visual, WordNet - visual_perception, visual_communication). To compensate for discrepancies between specifications for LAD data and user-system's data, LAD uses the mapping capability previously described. For example, when the SRM wants to know if a *carrot* is an *ingestible* item, LAD will access the mapping table to find *ingestible's* translation into WordNet categories. Finding that *ingestible* translates into *foodstuff* or *nutrient*,

LAD will search the WordNet database for *foodstuff* and/or *nutrient* as elements of the hierarchy for *carrot* and return a true or false answer to the SRM (see Appendix C for the complete WordNet to SRM mapping). In this manner, LAD transparently handles discrepancies in system specifications and its accessible source specifications.

Queries Requiring Multiple Databases

NLP systems often are designed with more than one knowledge base. These multiple knowledge bases enable complex reasoning about different "kinds" of data. For example, a system requiring corpora knowledge using LAD would get information from both the Brown and Carterette corpora. However, if they specified a specific corpus, then they would receive data from just one corpus.[5] For instance, frequency information can be determined in a number of ways, one being the word count of a word in a large corpus. LAD will have access to several corpora. These corpora differ in the type of information found in them; for instance, the Carterette Corpus contains samples of spoken language and the Brown Corpus contains samples of written language from a wide variety of sources. LAD provides multiple frequency information queries to enable users to chose specific databases from where they obtain frequency information, if they have a stronger preference for one corpus over the other. However, if no database is specified, LAD will use a default path to retrieve frequency information. Alternately, LAD could combine frequency infor-

5. Different corpora are gathered from different sources. Depending on the sources in the corpora, it can influence statistical data retrieved.

mation from several different sources using mapping information to decide if and how the information can reasonably be combined.

Another example of LAD requiring multiple databases is the searching methodology for verb case frames mentioned previously. If LAD cannot find a case frame for the verb it is processing, then it accesses information from WordNet that will enable it to continue its search in the verb case frame database.

Complicated Queries

At times, NLP systems require data that necessitates LAD to search in one or more databases before searching a final database to find the answer. For instance, in the case of a system requiring corpora statistics for a specific sense and part of speech information on a word, LAD would first need to access the WordNet database for the part of speech specified and then locate the specific sense of that word. Finally, LAD would need to call its corpora mapping function with the located word sense in order to retrieve the desired information. Another example of a complicated query would be one required for a computerized system with speech synthesis capabilities which relies on phonetic data to enable it to generate speech. LAD will be able to search the phonetic database for a given word and part of speech tag and return the pronunciation string. Phonetic information will be available for LAD through the phonetic database, but it will require a mapping function from a specific part of speech category for a word to the corresponding WordNet database (e.g., if the word is a noun, then it needs to access that word in the WordNet noun database). Then it needs to locate the intended word sense (based on semantic information it gets a word

sense, e.g., *lead* as a metal), and conclude its search by using a mapping utility that maps from WordNet to the pronunciations located in the phonetic database. This will enable LAD to accommodate different pronunciations of the same word (i.e., *lead* as in *He is the Project lead* vs. *lead* as in *I need lead for my pencil*).

Application Areas

LAD is designed to interact with multiple lexical databases in a transparent manner, so that the user-system treats LAD as a single dictionary. The resulting system will be a useful tool for various NLP-based AAC applications. Some applications that could benefit from LAD would be a system with a speech synthesizer needing pronunciation information, and a system with a syntactic-based word predictor using morphological information to predict correct verb forms. LAD will also be able to provide frequency information for systems designed to enable word and/or letter prediction. These systems rely on frequency information to determine appropriate choices for the next word/letter(s) of input. LAD is currently being tested with a semantic parser based on the reasoning principles used in Compansion (discussed in the Section, Application Domain: Augmentative and Alternative Communication, in Chapter 2). This application requires verb case frame information as well as semantic category information and is discussed in more detail in the next chapter. LAD can also provide the necessary information for other NLP-based systems, such as machine translation or syntactic parsers, as discussed at the end of Chapter 2.

Object-Oriented Design

The object-oriented paradigm is a set of theories, standards, and methods that together represent a way of organizing knowledge (Budd, 1991). Actions occur when a message is sent to an object, the object interprets the message and then performs some method in response to the message. All objects are instances of a class and if the same message is passed to multiple objects from the same class, they perform the same method. The sender of the message to the object does not need to know or even care to know how the action is accomplished, just that it is done (Budd, 1991). To take this into the realm of everyday living, if I call my local florist, Bill, and order a bouquet of red roses for my mother, I do not care about the details of how the florist actually fulfills my request; I just care that the order is filled. In this example, Bill, an instance of the class **florist**, gets a message, an order of a bouquet of red roses to be sent to Mary Mair, requiring him to use a method, create_red_rose_bouquet, a function providing actions to make up the bouquet of red roses, and another method, delivery, to deliver my order to my mother. In addition, I can be assured that if I call Bill to deliver a vase of pink carnations to my neighbor, Lorraine, Bill will recognize the different variables and use the appropriate methods to carry out this different request.

This notion of objects and classes is the basic building block of object-oriented programming. In order to build upon this to increase the power and flexibility of the paradigm the additional notion of inheritance is utilized. Classes can be organized into hierarchical structures that can inherit attributes and methods from the class (or classes) that it is derived from. To take this back into my previous example, Bill and I are both humans and

thus inherit a great deal of attributes and knowledge from the class **human**. I am also a Computer Scientist and thus have a great deal of knowledge from the class **Computer_Scientist**, while Bill is also a member of the **Florist** class and thus has a great deal of knowledge associated with the **Florist** class. In the **human** class there might be a function detailing the basic actions needed in order to create a bouquet of red roses, such as get a vase, fill it with water and place red roses in the vase. The **Florist** class needs more detailed information than this. Therefore they would have a method that overrides the basic method in the **human** class with one that goes into more detail, such as cutting the stems at an angle of 30 degrees while the stem is under water, placing the roses into a vase filled with specially treated nutritious water, and so on. With this brief discussion and example into what the object-oriented paradigm is, the following paragraph will describe the LAD object-oriented design as depicted in Figure 3.2.

LAD is based on the object-oriented paradigm. Its base class, **lad**, contains the virtual methods needed to perform its functions. Presently derived from the **lad** class are the **wordnet** class and the **case_frame** class. Their respective classes contain methods that require more detailed instructions than the basic methods in the **lad** class. For instance, the **is-a** function requires information from WordNet in order to return an answer of substance. The **lad** method for **is-a** only indicates that it returns 0 (false/NULL). On the other hand the **wordnet** class **is-a** method actually searches the WordNet databases and retrieves the **is-a** information associated with the specified word. The **case_frame** class **is-a** method knows how to handle **is-a** information about particular verb words. For instance, the query

break **is-a** *relational* would return false, since *break* is a *material* verb and not a *relational* verb.

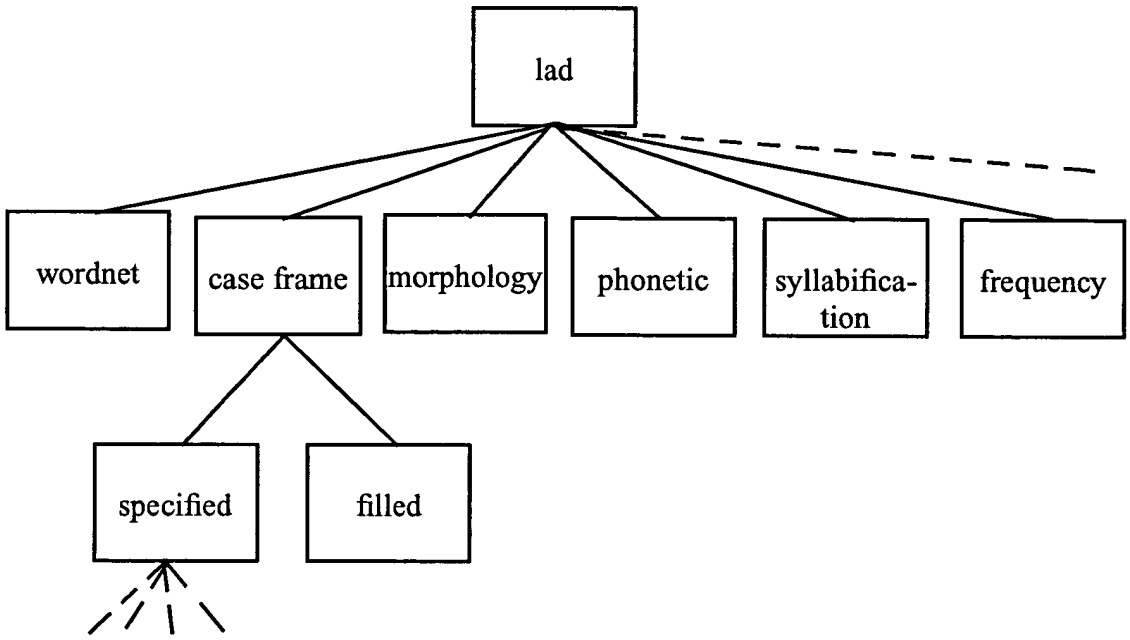


Figure 3.2 LAD Object-Oriented Class Hierarchy

As depicted in Figure 3.2, the level directly under the base class, **lad**, is the first level of derived classes that contain their own methods and data specifications. This enables LAD to inherit some functionality from the base class and overload those methods within their own classes that need to provide additional functionality. The level beneath the **case-frame** class depicts the two classes that form the case frame class (for more detailed information, see Appendix B); specified case frames, which hold the case and role

choices for specific verbs, and filled case frames, which hold the words that have filled a case and its preference rating. The dotted-lines below the **specified-case-frame** class indicate several derived classes for the different types of verbs (e.g., relational, attributive, oral, written). These classes specify to the system what the *toFillPref* ratings are on each case of a specified frame, and what categories (with their associated ratings) can fill each case. This hierarchial structure gives LAD the ability to grow and change with relative ease.

Extensibility

LAD has been designed using an object-oriented paradigm to increase its extensibility and robustness. By using this paradigm, information not currently available to LAD can be added easily without any major changes to the existing code. Instead, a new class can be created which knows how to handle the information accessible in the new lexical resources. If this class is derived from the LAD hierarchy, it will then be readily accessible to LAD. As an example, pronunciations are not available in WordNet or the case-frame secondary database which are currently accessible by LAD. By creating a pronunciation class, instances of pronunciation can be stored in a secondary database and accessed in a means similar to the way verbs with case frames are currently accessed in the case-frame database. The functionality of LAD is general enough to be used by many different derived modules.

A resource having functional capabilities not presently available in the LAD design, can be included in LAD with minimal code changes (e.g., add the new virtual func-

tions to the base **lad** class, derive the new class from **lad**, define the methods associated with this class, and add calling capabilities to the LAD driver in order to initiate the newly declared and defined functions). In this manner, LAD not only has the ability to increase its lexical and linguistic resources, but it can incorporate new design features with a minimal amount of effort. This is a notable difference and improvement over the design of the system developed by Knight and Luk (1994) mentioned in Chapter 2. They merge their five lexical resources together, and would require substantial alterations to the existing code in order to incorporate new resources, if they were desired.

Implementation Status

LAD has not been fully implemented, as we felt it was important to complete a full analysis of LAD's functional design, implementing the WordNet and case frame classes with enough functionality so that testing and then evaluation could be performed. Currently implemented are the functions that perform the is-a, list-semantic-categories, has-a and case frame queries (for more details on LAD's functions, see Appendix A) which are needed by the Semantic Reasoning Module (SRM). The following chapter will go into the SRM in more detail.

Chapter 4

SEMANTIC REASONING MODULE

Motivation

LAD was designed to be a useful lexical resource for multiple applications. In order to test LAD's capabilities, the Semantic Reasoning Module[1] (SRM) was designed and implemented. The SRM takes as input uninflected content words intended as a telegraphic message.[2] The SRM returns a list of all the possible filled case frames which capture possible intended messages associated with the input words. Each filled case frame indicates which role each of the words play with regard to the verb, and has a "case rating" associated with it that can be used to compare the relative goodness of the various generated filled case frames. In order to accomplish this, the SRM obtains case frame and semantic information from LAD. The following sections will describe certain requirements of the SRM and how LAD is able to satisfy them (for more detailed SRM function specification, see Appendix B).

---

1. The SRM was based on principles developed in Compansion, but implemented in C++, instead of Lisp, to test out the theoretical and functional design of LAD.

2. In the input, the verb is identified and the other words are limited to be nouns which play some role with respect to the verb.

Case Frames and Semantic Reasoning

The SRM uses two case frame representations to perform semantic reasoning. The first type of case frame is the specified verb case frame, previously discussed in Chapter 2. The second is the filled case frame, described in more detail in the section on Semantic Output in this chapter, which will be used to generate syntactically and semantically correct sentences.

The SRM takes as input a verb and one or more uninflected nouns, which are intended to fit together into a well-formed sentence. The system's goal is to determine the proper roles of the content words with respect to the verb, and to generate filled case frames that reflect all possible combinations of these roles. The first interaction between LAD and the SRM consists of the extraction of a verb case frame for the input verb. The information contained within the case frame specifies what roles can be filled, how important it is to fill each of these roles, and the kinds of words that can be used to fill each role. This case frame provides semantic expectations for the remaining words of input. The next interaction between LAD and the SRM is to determine what semantic categories the words of input can fill. In some cases these semantic categories must be mapped by LAD onto categories appropriate for the verb frame by LAD's mapping mechanism (as discussed in Chapter 3). Using the information retrieved by LAD, the SRM can generate all possible filled case frames and present them to the user. The filled case frames (along with their goodness rating) is the SRM output. These could then be taken by another component to generate syntactically and semantically correct sentences for the words of input,

and could be returned to the user for feedback as to which generated sentence best matches their intended meaning.

Semantic Categories

The semantic categories used by the Semantic Reasoning Module capture information that can be associated with words. This information provides distinctions between classes (e.g., animate vs. inanimate) and is motivated by the roles that *objects* can play with respect to various verbs. These semantic categories are the same categories employed by the Compansion system (see Appendix C).

Preferences

The job of the SRM is to determine likely sentence meaning. For the SRM, this sentence meaning is captured in a case frame representation based on work by Fillmore (1977). In this representation, the verb is central and the nouns are said to play a small number of roles with respect to the verb. The roles used include: AGEXP (agent/experiencer) is the object doing the action. For us, the AGEXP does not necessarily imply intentionality such as in predicate adjective sentences (e.g., *John* is the AGEXP in *John is happy*). THEME is the object being acted upon, while INSTR is the object or tool used in performing the action of the verb. GOAL can be thought of as a receiver, which is not to be confused with the BENEF (beneficiary) of the action. For example, in *John gave a book to Mary for Jane*, *Mary* is the GOAL while *Jane* is the BENEF. We also have a LOC case which captures the location in which the situation is taking place (this case may be further

decomposed into TO-LOC, FROM-LOC, and AT-LOC), and TIME which captures time information (this case may also be further decomposed).

The final output of the SRM creates a number of filled and rated case frames which place the nouns of the input in all of their reasonable roles. So, for example, the input, *mary like car*, generates two different filled case frames (one with a rating of 18 and the other with a rating 9):

```
The verb is like
The total filled case rating is 18
The agexp->caseName is mary the caseFiller category is
                       human with a rating of 12
The theme->caseName is car the caseFiller category is
                       human with a rating of 6


The verb is like
The total filled case rating is 9
The theme->caseName is car the caseFiller category is
                       human with a rating of 6
The benef->caseName is mary the caseFiller category is
                       human with a rating of 3
```

The SRM must have the ability to reason about the most likely way that the given words can fit into a case frame. In order to do this, a set of preference ratings are associated with each verb case frame which indicate possible ways of completing a case frame. In addition, these preferences are used to rank the filled case frames against each other. There are three different kinds of semantic case preferences: case filler preferences, case importance preferences, and higher-order case preferences. The case filler preferences are found in many NLP case-based systems and indicate preferences for the semantic categories that are most reasonable for filling out a given case (e.g., animate agents are preferred for most

verbs), the case importance preferences indicate which cases are most important to fill (e.g., the agent case is generally more important to fill than the beneficiary case), and higher-order case preferences (which capture interactions between the way various cases are filled out). Preference ratings fall on a 1-4 scale: 4 signifies a high preference, while 1 signifies while acceptable, it is only appropriate in special cases.

The case filler preferences are used in other semantic representations to indicate preferred filled roles of a particular verb case frame and are based on the case filler preferences described in Preference Semantics (Wilks, 1975). The kinds of objects that could fill a particular role are indicated, along with a rating for each possible role filler type. For example, the preference for filling the BENEF case for *break* is: ((human 3) (organization 2) (animate 2)). This indicates that given the choice, a human should fill this role, but an organization or an animate object are also reasonable alternatives.

Case importance preference ratings indicate what *cases* are more important to fill for a particular verb. For example, with the case frame shown in Figure 4.1 for *like*, it is more likely that the role of AGEXP will be filled than any of the other cases. To indicate this, a higher value (four) is given as the preference of filling the AGEXP case, while lower values (one or two) are given as the preference for filling the other cases.

The higher-order case preferences are used to compensate for exceptions-to-the-rule situations. For instance, if a non-human animate (e.g., *dog*) fills the AGEXP role for the verb *eat*, it is highly unlikely that an INSTR is being used by the *dog* to do the *eating*. Yet if a *human* fills the AGEXP role, this is very reasonable. The idea is to use these high-

er-order case preferences to compensate for this kind of situation by, in essence, lowering the case importance preference rating.

```
verb - like
agexp [toFillPref 4] [[human 3] [organization 2] [animate 1]]
theme [toFillPref 2] [object 3]
instr [toFillPref 1] [[cognitive 3] [tool 1]]
benef [toFillPref 1] [[human 3] [organization 2] [animate 2]]
loc   [toFillPref 1] [place 4]
timee [toFillPref 1] [time 4]
```

Figure 4.1 Specified Case Frame for the Verb LIKE

These preferences are captured in a set of verb frames which are stored in one of the lexical resources available to LAD, which contain functions for retrieving the verb frames. For example, Figure 4.1 shows the verb case frame for *like*, the case filler preferences (indicated by *toFillPref* after each role) indicate a very high preference for filling the AGEXP case, and a preference for filling the THEME case over the other cases. The case filler preferences for each role are captured in the list following each *toFillPref*, they indicate that a *human* is preferred for the AGEXP role, however an *organization* or *animate* is also acceptable, and that an *object* is needed to fill the THEME role. Thus, given the input *like mary car*, *mary* can fill the roles of the AGEXP or BENEF (though the role of *agexp* is preferred over the BENEF) and *car* can fill the role of THEME.

SRM and LAD (Knowledge Representations)

As was discussed in Chapter 3, LAD relies on WordNet to retrieve semantic categories of words. One problem is that a particular application (e.g., SRM) may use a set of semantic categories which do not exactly match the categories that WordNet uses. This is in fact the case with the SRM since it is based on the case frames developed for the Compansion system (see Appendix C for the SRM's semantic categories). For instance, the SRM refers to a class *inanimate*, although WordNet does not classify any words as *inanimate*, but instead classifies them as *inanimate_objects*. To compensate for this, LAD uses its mapping function to map to and from the WordNet and the SRM's semantic categories. Thus it maps *inanimate* to *inanimate_objects* and allows the SRM to function as though WordNet stored the semantic information identically to its needs.

Secondary Verb Frames vs. WordNet Verb Frames

The verb case frame database has a small number of entries (88 to be exact), while the SRM needs LAD to return verb case frames for a much larger number of verbs. LAD accomplishes this task by not only searching the verb case frame database for case frames, but if that search is empty, proceeding to generate a list of synonyms for the verb and searching the secondary database for a synonymous entry. If an entry is found for one of the verb's synonyms, then that case frame is returned. Part of our future work will investigate how the system might recover if no synonymous verb has an entry. For example, one approach would be to generate a case frame from the WordNet verb frame data (see Appendix D). These case frames will not be as complete as the other case frames (i.e., they

may not contain as many semantic categories in their preferences). However they would allow the SRM to continue reasonable processing. Again, as in the case of the semantic category mapping, the verb case frame search is done in a transparent manner.

Semantic Output

The output from the SRM is currently a list of filled case frames, indicating what role each word of input should play for a particular possible sentence generation, with a total rating on the goodness of the sentence. Figure 4.2 contains a list of the filled case frames for the input sequence *break mary window hammer.*

The preference rating for a sentence is the summation of the ratings on the filled roles in the filled case frame (gotten by multiplying the to fill preference and the case filler preferences). The preference ratings for the filled frames shown in Figure 4.2 are 21, 12, and 12 respectively. From this, the preferred interpretation comes from the first frame which might be generated as *Mary broke the window with the hammer.* The generator might return all possible generations in such a manner that the sentence with the highest preference rating is returned first, the next highest rated sentence next, and so forth. The user could then select the preferred sentence from those generated, with the most likely generation being at the top of the list.

```
The verb is break
The total filled case rating is 21
The agexp->caseName is mary the caseFiller category is
                       human with a rating of 12
The theme->caseName is window the caseFiller category
                       is object with a rating of 3
The instr->caseName is hammer the caseFiller category
                       is tool with a rating of 6

The verb is break
The total filled case rating is 12
The theme->caseName is window the caseFiller category
                       is object with a rating of 3
The instr->caseName is hammer the caseFiller category
                       is tool with a rating of 6
The goal->caseName is mary the caseFiller category is
                       human with a rating of 3

The verb is break
The total filled case rating is 12
The theme->caseName is window the caseFiller category
                       is object with a rating of 3
The instr->caseName is hammer the caseFiller category
                       is tool with a rating of 6
The benef->caseName is mary the caseFiller category is
                       human with a rating of 3
```

Figure 4.2 List of Filled Case Frames

SRM Processing

The SRM can perform very complex processing since many of the input words may appropriately fill several roles. The SRM is required to ensure that: 1) each word of input appears in some role in every generated filled case frame; 2) that no two words fill the same role in a filled case frame; and 3) that every filled case frame has a unique assignment of input words to roles. To handle this complex processing, the SRM uses the follow-

ing algorithm shown in Figures 4.3 and 4.4. At the start of processing, the algorithm has a list of words, each with an associated rolelist. The rolelist indicates each role a word may play (according to the semantic instructions captured in the case frame) and a preference strength associated with each role that indicates how much that word filling that role would add to the filled frame's overall rating. For clarity, the "failure points" have been left out of the algorithm. For instance, if two words of input could each only fill the same role, the algorithm would not be able to generate any filled case frames. The algorithm has been separated into two figures to depict the two main loops of processing done by the SRM.

1. Check for all words that can only fill one role and place them on the mandatory rolelist.

2. Call **ridConflicts** to eliminate the roles filled in Step 1 from the remaining word's rolelists.

3. Go back to Step 1 until there are no conflicting roles or words with only one possible role to fill remaining.

4. If there are no words remaining with multiple possible roles, then generate a filled case frame using the words on the mandatory rolelist, otherwise go to Step 5 to handle the recursion required for further processing.

Figure 4.3 Algorithm for the Generation of Filled Case Frames (Part 1)

At this point, either the processing is completed (in the case where all the words of input can only fill one role once conflicts are eliminated), or else more complex processing needs to proceed (in the case where there are words that can fill multiple roles).

5. Take the first word with multiple possible roles off of the multiple role wordlist.

6. Pop a role off of this word's rolelist and put it on a temporary list.

7. If there are more words on the multiple role wordlist, go to Step 8, otherwise generate a filled case frame using the mandatory list and the temporary list. Put this frame on the list of filled case frames to be returned. If there are no more roles on this word's rolelist, then return the list of filled frames, otherwise go back to Step 6.

8. Check the rolelists of the next word on the multiple role wordlist to see if its first role conflicts with the roles on the temporary list, if it does, skip this role and move to the next role. Repeat Step 8 until a role is reached that does not conflict with any roles on the temporary list.

9. If there are more words on the multiple role wordlist, go back to Step 8, else if there are no more roles on this word's rolelist, add the current role from this word to the frames on the partially filled framelist, otherwise go to Step 10.

10. Generate a filled case frame for each entry remaining on this word's rolelist, and put the frame on a partially filled framelist.

Figure 4.4 Algorithm for the Generation of Filled Case Frames (Part 2)

This algorithm ensures each word of input is in every generated filled case frame without any words conflicting over a role, and that every generated frame is unique. The next chapter concludes this thesis with an analysis and evaluation of LAD, its design, implementation, testing results and its present resource limitations, and a discussion on future work.

Chapter 5

CONCLUSIONS

The Language Access Database was designed to give a user access to several on-line lexical and linguistic sources in a unified manner thus providing various different software applications with a convenient source of syntactic, semantic, and other lexical information. This information will enhance the capabilities of systems which attempt to understand natural language, to generate semantically and/or syntactically correct sentences, to produce speech synthesized language, and so on. LAD was designed using the principles of the object-oriented paradigm to allow for flexibility and extensibility.

This final chapter examines the goals and accomplishments of LAD, an analysis of the LAD system (e.g., testing and evaluations, resource weaknesses, comparison of output from the SRM's LAD enriched system and the Compansion system), a discussion of the results of LAD, and concludes with a look into the future directions for LAD.

Accomplishments

The completed LAD design was based on providing traditional dictionary functionality as well as enhanced capabilities to incorporate the semantic and syntactic needs of NLP-based systems. The current implementation encompasses enough of the LAD functionality to be tested and evaluated before a complete implementation is done. In its

current form, LAD has access to the WordNet database and a verb case frame database. The Semantic Reasoning Module, used as a testing and validation system, was designed with principles developed in the Compansion system.

LAD Implementation Status

LAD has been fully designed, but not fully implemented. The reason for this is that LAD has been designed as a fully extensible system. Thus, as new applications need to use LAD, everything is in place to add access to any additional lexical resources they might require. The completed portion of LAD is sufficient for testing its usefulness in the implementation of the SRM. The functionality required for this implementation utilize all of LAD's major design components (i.e., multiple database access, an application-specific database in the case frame database, mapping information required in mapping data from one database terminology to another, and sophisticated access to WordNet's semantic information). Thus the implemented portion of LAD provides significant access to needed information that was not previously available. In addition, the testing of LAD using the SRM validates the design principles upon which LAD is based.

Now that the first stage of testing is completed, the results warrant further testing by using LAD in other applications. Currently implemented are the functions needed for the SRM processing. These perform the is-a, list-semantic-categories, has-a and case frame queries. The LAD implementation will proceed to incorporate more lexical resources, as its design intended, and more of the functions as applications and lexical databases warrant.

WordNet Usage

WordNet has proven to be an excellent choice as the main lexical database due to its broad coverage of words and sense information. It has limitations in certain areas of its lexicon, the most notable being proper names. However this will be compensated for by adding a mapping function using the Moby Words II proper names of people and proper names of places files to their equivalent expressions in WordNet. For instance, the mapping mechanism can be used to map male names like *John* to *Tom* (which is included in WordNet), and to map names of cities to equivalent cities (i.e., taking care to map seaports to seaports, major urban areas to other major urban areas, etc.). Moby Words II contains 21,986 names, including 4,946 commonly given female names and 3,897 commonly given male names in English speaking countries, and 10,196 places in the United States. This addition to WordNet will improve one of its weaknesses, although there is still a lack of proper names of products and businesses (e.g., Ford, Chevy, IBM, duPont). This final weakness could be corrected with a flat-file database of names of companies and their products, if needed or desired.

SRM and LAD Results

The results of the SRM implementation using LAD were very supportive of further work with LAD, and with other applications that will make use of it. There were a total of 32 input test strings processed by the SRM, and the expected results[1] were achieved from

1. The SRM produced a list of filled case frames which comply with its complex processing requirements (see Chapter 4, Section SRM Processing), each frame is semantically logical, and at least one of the filled case frames could be used to generate a valid sentence.

29 of these test strings. The three input test strings which did not work in the SRM were due to failures in the mapping function used to map the semantic categories of the SRM to the names found in WordNet. For example, WordNet does not have a definition or sense of the word *bone* as in a bone a dog would eat. The SRM also uses the category name *instrument* and WordNet uses *instrumentality* for the word *fork*, and the SRM uses the category name *time* and WordNet uses *season* and *time_of_year* for the word *summer*. Presumably, the mappings in the mapping table (see Appendix C) could be extended to handle these cases.

Some of the test strings included in the SRM test were:

```
go mary tom store
break mary window hammer
break mary window hammer tom
break mary hammer rock tom
ask mary question
hit tom mary dog
buy mary book yesterday
eat mary slice bread
break mary thermostat
utter mary sentence
steal mary bread
go mary restaurant yesterday
fly mary europe
order mary hamburger tom
write mary paper chemistry
go mary beijing
eat mary hamburger fork
remember mary swimming summer
eat mary dog bone
```

In the 29 successful test strings, LAD enabled the SRM to produce complete lists of filled case frames represented by the words of input, where each frame contains all of

the words and every frame has a different pattern of words for the cases they fill. For example, consider the input string *break mary window hammer.* The cases that *mary* can fill are AGEXP (with the semantic category of human and a rating of 12), GOAL (with the semantic category of human and a rating of 3), and BENEF (with the semantic category of human and a rating of 3). The case that *window* can fill is THEME (with the semantic category fragile and a rating of 1). Finally, the cases that *hammer* can fill are THEME (with the semantic category object and a rating of 3), and INSTR (with the semantic category tool and a rating of 6). You will notice that *mary* can fill three different cases, while *window* can only fill the THEME case, and *hammer* will only be allowed to fill the INSTR case, as the THEME case will always be filled by *window.* Therefore the results from the SRM with these words of input should (and do) yield the correct filled case frames (as shown in the next section).

The results of the SRM implementation using LAD were very good and show that LAD has a promising future for providing various applications with semantic knowledge. The mapping function needs to be further tested and refined to account for discrepancies, as became evident during the testing session. LAD has thus proven to be a useful tool for providing semantic knowledge.

SRM Performance vs. Compansion Performance

The SRM does not have the full functionality of Compansion because it cannot generate sentences, nor handle certain exceptional cases. However, a comparison of the SRM-generated filled case frames with the sentences generated from Compansion is feasi-

ble by using the same (or equivalent) test strings. The following is an example of the SRM testing results:

```
input string: break mary window hammer


The verb is break
The total filled case rating is 30
The agexp->caseName is mary the caseFiller category is
                       human with a rating of 12
The theme->caseName is window the caseFiller category is
                       fragile with a rating of 12
The instr->caseName is hammer the caseFiller category is
                       tool with a rating of 6


The verb is break
The total filled case rating is 21
The theme->caseName is window the caseFiller category is
                       fragile with a rating of 12
The instr->caseName is hammer the caseFiller category is
                       tool with a rating of 6
The goal->caseName is mary the caseFiller category is
                       human with a rating of 3


The verb is break
The total filled case rating is 21
The theme->caseName is window the caseFiller category is
                       fragile with a rating of 12
The instr->caseName is hammer the caseFiller category is
                       tool with a rating of 6
The benef->caseName is mary the caseFiller category is
                       human with a rating of 3
```

Compansion has a generator which generates sentences from the filled case frames its semantic parser builds. The generator may not be able to generate semantically and syntactically correct sentences from all of the frames, so the output from Compansion is

only the sentences it is able to generate. The following is the Compansion output for the same test string as in the previous SRM example:

```
input string: mary break window hammer
output--> Mary breaks the window with the hammer.
```

The first filled case frame from the SRM would be used by a generator to create the same sentence as the output from Compansion above. The following two filled case frames might be used to generate the sentence "The window was broken with the hammer for Mary." It is important to note that Compansion takes word order into account, so the previous sentence could not be generated from the ordered input string *mary break window hammer.* What is encouraging from these results is that for input strings which do not require the extra exception processing Compansion provides, similar results were produced.

When Compansion was tested on the same 32 strings that were used to test the SRM the results were that it could successfully process 15 of the 29 strings that the SRM could process and one of the strings that the SRM could not process (*mary eat pizza fork*). This difference in results is mainly due to the fact that the SRM has a larger vocabulary than Compansion, because the SRM obtains its word knowledge from LAD. While Compansion is limited to a vocabulary of over 1,000 words, the SRM has access to the over 95,000 words in WordNet and once the proper names mapping is completed, it will have an additional 21,986 proper names access from the MobyWords data files. Some of the test input strings that the SRM could handle and Compansion could not were:

```
break mary hammer rock tom
break mary hammer rock
write mary paper chemistry
go mary beijing
eat mary slice bread
eat mary dinner tom
break mary thermostat
utter mary sentence
steal mary bread
corrupt mary morals
build mary software
design mary software
go mary restaurant yesterday
fly mary europe
```

In all of the cases above, except for the first three, Compansion's failure was due to lack of access to semantic knowledge about one or more of the words in the input string. In the first three cases, the failure may have been due to a lack of semantic knowledge or due to the generator not being able to formulate a sentence which captured the semantic representation produced by Compansion's semantic parser. For the two cases which failed for both Compansion and the SRM, it was because of vocabulary limitations of Compansion and mapping failures in LAD.

Discussion

LAD has proven to be a valuable semantic tool, enabling the NLP-based semantic parser, SRM, by providing the knowledge it requires to process a large number of telegraphic inputs. The vocabulary size that LAD provides is significantly larger than the prototype Compansion system, and this greatly increases the variety of input strings the SRM can process. LAD now needs to be provided with access to additional databases and addi-

tional capabilities of the WordNet database so that it can be integrated with other applications. The initial results on LAD are very encouraging, and validate the need for future work.

Future Work

As mentioned previously, LAD has been fully designed, but only partially implemented. A great deal of the future work lies in completing the implementation of LAD, linking in more data sources, and implementing the undefined methods. Some specific features to be included in the future work will be detailed in the following paragraphs.

Currently, LAD retrieves a verb case frame from a secondary database by default. On occasions where the verb is not represented in the secondary database, a case frame is generated by first searching synonyms of the verb from WordNet (e.g., *crush*) and then checking in the secondary database for these synonyms (e.g., *beat, defeat, whip, trounce, vanquish, overcome, fragment, break, separate*). Presently, if that search still fails, processing is halted. However, in the future we would like to be able to generate a case frame based on the WordNet verb frame (see Appendix D). These verb frames are very basic and lack detail (e.g., Somebody ---s something), but could be used to build a minimal specified case frame from which further processing could continue.

The specified and filled case frames presently accessed by LAD are useful semantic tools. In the future it would be beneficial to have verb case frames that contain syntactic information as well, based on Beth Levin's work on diathesis alternations and verb classes. Using her verb classes, it is hoped that verb case frames can be developed that would

incorporate syntactic information as well as semantic meaning. This would improve the syntactic generation of sentences from the filled case frames. The initial stages of this work is discussed in (Zickus, 1994).

A number of enhancements are being planned that will increase the ultimate utility of LAD, including a compiler that will produce a more compact version of the database based on a specification of words. This would reduce the overall memory and disk space requirements when used in a practical system. For instance, one could imagine generating a database containing just pronunciation information for some specific list of input words, or some list of words which occur at or above a specified frequency in the Brown corpus. It is anticipated that this compiler could be used to generate a lexicon of specific information for an application that might run on a PC or AAC device with limited memory. In addition, while LAD is intended to be primarily used by programmers, it will also be necessary for non-technical people to enter new information into the system, and for this, a front-end program will be developed to facilitate this process.

BIBLIOGRAPHY

Albacete, P. L., Chang, S. K., Polese, G., and Baker, B. (1994). Iconic Language Design for People with Significant Speech and Multiple Impairments. *ASSETS '94 - The First Annual ACM Conference on Assistive Technologies*, 23-30.

Allen, B. P. (1994). Case-Based Reasoning: Business Applications. *Communications of the ACM* (Vol. 37, No. 3), 40-42.

Allen, J. (1987). *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park.

Alm, N., Todman, J., Elder, L., and Newell, A. F. (1993). Computer Aided Conversation for Severely Physically Impaired Non-Speaking People. *Human Factors in Computing Systems: INTERCHI'93 Conference Proceedings*, ACM Press, New York, 236-241.

American Heritage Dictionary (1985). 2nd College Edition. Houghton Mifflin Company Publishers, Boston.

Anson, D. (1993). The Effect of Word Prediction on Typing Speed. *The American Journal of Occupational Therapy* (Vol. 47, No. 11), 1039-1042.

Archer, L. (1977). Blissymbolics: A Nonverbal Communication System. *Journal of Speech and Hearing Disorders* (Vol. 42), 568-579.

Baker, B. (1982). Minspeak: A Semantic Compaction System that Makes Self Expression Easier for Communicatively Disabled Individuals. *Byte* (Vol. 7, No. 9), 186-202.

Berwick, R. C., Jones, D., Cho, F., Kahn, Z., Kohl, K., Radhakrishnan, A., Sauerland, U., and Ulicny, B. (1994). Issues in Modern Lexical Theory: the (E)VCA Project. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 47-61.

Budd, T. (1991). *An Introduction to Object-Oriented Programming*. Addison-Wesley Publishing Company, New York.

Byrd, R. J. (1989). Large Scale Cooperation on Large Scale Lexical Acquisition. *Workshop on Lexical Acquisition, IJCAI*. Detroit.

Chang, S. K., Costagliola, G., Orefice, S., Polese, G., and Baker, B. R. (1992). A Methodology for Iconic Sentences for Augmentative Communication. *Proceedings of the 1992 IEEE Workshop on Visual Languages*, 110-116.

Chang, S. K., Orefice, S., Polese, G., and Baker, B. R. (1993). Deriving the Meaning of Iconic Language Design with Application to Augmentative Communication. *Proceedings of the 1993 IEEE Workshop on Visual Languages*, 267-274.

Chen, Z. and Huang, H. (1994). Lexical Knowledge Organization in Machine Translation. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 108-111.

Church, K. W. (1994). Just-In-Time Lexicons. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 1-10.

Demasco, P. W., and McCoy, K. F. (1992). Generating Text From Compressed Input: An Intelligent Interface for People with Severe Motor Impairments. *Communications of the ACM*, (Vol. 35, No. 5), 68-78.

Demasco, P., and Mineo, B. (1995). AAC Technology: Next Year and Next Decade. *Presentation at the Pennsylvania Speech Hearing Language Association*, Valley Forge, PA. March 1995.

Demacso, P., Newell, A. F., and Arnott, J. L. (1994). The Application of Spatialization and Spatial Metaphor to Augmentative and Alternative Communication. *ASSETS '94 - The First Annual ACM Conference on Assistive Technologies*, 31-38.

Fass, D., and Wilks, Y. (1983). Preference Semantics, Ill-Formedness, and Metaphor. *American Journal of Computational Linguistics* (Vol. 9, Nos. 3-4), 188-196.

Fellbaum, C. (1993). English Verbs as a Semantic Net. *CSL Report 43*, July 1990, Revised March 1993.

Fellbaum, C., Gross, D., and Miller, K. (1993). Adjectives in WordNet. *CSL Report 43*, July 1990, Revised March 1993.

Fillmore, C. J. (1977). The Case for Case Reopened. *Syntax and Semantics VIII Grammatical Relations*, P. Cole and J. M. Sadock (eds.), Academic Press, 59-81.

Foulds, R. A. (1980). Communication Rates for Nonspeech Expression as a Function of Manual Tasks and Linguistic Constraints. *Proceedings of International Conference on Rehabilitation Engineering - Toronto*, 83-87.

Granger, R. H. (1983). The NOMAD System: Expectation-Based Detection and Correction of Errors During Understanding of Syntactically and Semantically Ill-Formed Text. *American Journal of Computational Linguistics* (Vol. 9, Nos. 3-4), 188-196.

Gros, J., Zganec, M., Mihelic, F., and Pavesic, N. (1994). A Lexicon for Automatic Speech Recognition and Understanding. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 186-191.

Gu, Y. (1994). Some Theoretical Considerations on the Lexicon with Reference to the Minimalist Program. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 62-65.

Hamilton, K. (1994). Predictive Letter Scanner for Augmentative Communication. *Proceedings of the RESNA '94 Annual Conference*, M. Binion (ed.), RESNA Press, Arlington, VA, 121-123.

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., and Slocum, J. (1978). Developing a Natural Language Interface to Complex Data. *Systems* (036-5915), ACM Press, 563-584.

Hogan, C., and Levin, L. S. (1994). Data Sparseness and the Acquisition of Syntax-Semantic Mappings from Corpora. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 153-159.

Ide, N., and Veronis, J. (1994). Machine-Readable Dictionaries: What Have We Learned, Where Do We Go? *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 137-146.

Jackendoff, R. (1978). Grammar as Evidence for Conceptual Structure. *Linguistic Theory and Psychological Reality*, M. Halle, J. Bresman, and G.Miller (eds.), MIT Press, Cambridge, 201-228.

Jensen, K., Heidorn, G. E., Miller, L. A., and Ravin, Y. (1983). Parse Fitting and Prose Fixing: Getting a Hold on Ill-Formedness. *American Journal of Computational Linguistics* (Vol. 9, Nos. 3-4), 147-160.

John, B. E., and Morris, J. H. (1993). HCI in the School of Computer Science at Carnegie Mellon University. *Human Factors in Computing Systems: INTERCHI'93 Conference Proceedings*, ACM Press, New York, 49-50.

Johnson, G. J. (1994). Of Metaphor and the Difficulty of Computer Discourse. *Communications of the ACM* (Vol. 37, No. 12), 97-102.

Jubak, J. (1992). *In the Image of the Brain: Breaking the Barrier Between the Human Mind and Intelligent Machines*. Little Brown and Company, Boston.

Koester, H. H., and Levine, S. P. (1994). Validation of a Keystroke-Level Model for a Text Entry System Used by People with Disabilities. *ASSETS '94 - The First Annual ACM Conference on Assistive Technologies*, 115-122.

Knight, K., and Luk, S. K. (1994). Building a Large-Scale Knowledge Base for Machine Translation. *Proceedings of the 12th National Conference on Artificial Intelligence*. Part 1 (of 2), Seattle, WA, 773-778.

Kraat, A. W. (1990). Augmentative and Alternative Communication: Does It Have a Future in Aphasia Rehabilitation? *Aphasiology* (0268-7838 Vol. 4, No. 4), 321-338.

Levin, B. (1992). A Preliminary Analysis of (De)Causative Verbs in English. *Workshop on the Acquisition of the Lexicon* - University of Pennsylvania, January 1992, 1-36.

Levin, B. (1993). *English Verb Classes and Alternations: A Preliminary Investigation*, The University of Chicago Press.

Levin, B., and Hovav, M. R. (1992). The Lexical Semantics of Verbs of Motion: the Perspective from Unaccusativity*. *Thematic Structure Its Role in Grammar*, I.M. Roca (ed.). Foris Publications, New York, 247-269.

Levin, B., and Pinker, S. (1992). Introduction. *Lexical & Conceptual Semantics*, B. Levin and S. Pinker (eds.), Blackwell Publishers, Cambridge, 1-6.

Light, J. (1988). Augmentative Communication: State of the Art in North America. *ICAART88 - Montreal*, 536-540.

Macleod, C., Grishman, R., and Meyers, A. (1994). Developing Multiply Tagged Corpora for Lexical Research. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 11-22.

Magnusson, J.H., and Briem, S. (1994). Application of Isbliss and Blissgrammar Symbolic Processing Programs. *Contribution to the International Conference Beyond Normalization Towards One Society for All.* Reykjavik, Iceland, 1-11.

McCoy, K. F., and Demasco, P. W. (1995). Some Applications of Natural Language Processing to the Field of Augmentative and Alternative Communication. *The Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95*, Montreal, August 1995. Submitted.

McCoy, K. F., Demasco, P. W., Jones, M. A., Pennington, C. A., Vanderheyden, P. B., and Zickus, W. M. (1994A). A Communication Tool for People with Disabilities: Lexical Semantics for Filling in the Pieces. *ASSETS '94 - The First Annual ACM Conference on Assistive Technologies*, 107-114

McCoy, K. F., McKnitt, W. M., Peischl, D. M., Pennington, C. A., Vanderheyden, P. B., and Demasco, P. W. (1994B). AAC-User Therapist Interactions: Preliminary Linguistic Observations and Implications for Compansion. *Proceedings of the RESNA '94 Annual Conference*, M. Binion (ed.), RESNA Press, Arlington, VA, 129-131.

McHale, M. L., and Crowter, J. J. (1994). Constructing A Lexicon From A Machine Readable Dictionary. *Army Rome Laboratory Technical Report*, #RL-TR-94-178, Rome Laboratory, Griffiss AFB, New York.

McKevitt, P., and Guo, C. (1994). From Chinese Rooms to Irish Rooms: Perspectives on Language and Perception. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 160-173.

Miller, G. A. (1993). Nouns in WordNet: A Lexical Inheritance System. *CSL Report 43*, July 1990, Revised March 1993.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). Introduction to WordNet: An On-Line Lexical Database. *CSL Report 43*, July 1990, Revised March 1993.

Miller, G. A., and Fellbaum, C. (1992). Semantic Networks of English. *Lexical & Conceptual Semantics*, B. Levin and S. Pinker (eds.), Blackwell Publishers, Cambridge, 197-229.

Mineo, B., Demasco, P., Gray, J., and Bender, R. (1994). Systematic Assessment of Picture-Based Language Performance Via Computer. *Proceedings of the 1994 ISAAC Conference*, Maastricht, The Netherlands: IRV, 111-113.

Morris, C., Newell, A., Booth, L., Ricketts, I., and Arnott, J. (1992). Syntax PAL: A System to Improve the Written Syntax of Language-Impaired Users. *Assistive Technology* (Vol. 4, No. 2), RESNA Press, Arlington, VA, 51-59.

Newell, A. F. (1987). How Can We Develop Better Communication Aids? *AAC Augmentative and Alternative Communication* (0743-4618/87/0301-0036), 36-40.

Palmer, M. (1990). Customizing Verb Definitions for Specific Semantic Domains. *Machine Translation 5*, Kluwer Academic Publishers, 5-30.

Palmer, M. and Polguere, A. (1994). A Lexical and Conceptual Analysis of BREAK: A Computation Perspective. *Computational Lexical Semantics*, P. Saint-Dizier and E. Viegas (eds.), Cambridge University Press, to appear in 1994.

Pustejovsky, J. (1991). The Generative Lexicon. *Computational Linguistics* (Vol. 17, No. 4), 409-441.

Rich, E., and Knight, K. (1991). *Artificial Intelligence* (second edition). McGraw-Hill, Inc. New York.

Shneiderman, B. (1993). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (second edition). Addison-Wesley Publishing Company, New York.

Small, S., and Rieger, C. (1982). Parsing and Comprehending with Word Experts (A Theory and its Realization). *Strategies for Natural Language Processing*. W.G. Lehnert and M.H. Ringle (eds.), Lawrence Erlbaum Associates publishers, Hillsdale, NJ, 89-147.

Stum, G. M., and Demasco, P. (1992). Flexible Abbreviation Expansion. *Proceedings of the RESNA International '92 Conference*, J. J. Presperin (ed.). Washington, D.C. RESNA Press, Arlington, VA, 371-373.

Suri, L. Z., and McCoy, K. F. (1993). Correcting Discourse-Level Errors in a CALL System for Second Language Learners. *Technical Report 94-02*, Department of Computer and Information Sciences, University of Delaware, Newark, DE.

Sutcliffe, R. F. E., O'Sullivan, D., Sharkey, N. E., Vossen, P., Slator, B. E. A., McElligott, A., and Bennis, L. (1994). A Psychometric Performance Metric for Semantic Lexicons. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 75-88.

Tin, E., and Akman, V. (1994). Computational Situation Theory. *Sigart Bulletin* (Vol. 5, No. 4), ACM Press, New York, 4-18.

Vanderheiden, G. C. (1984A). Augmentative Communication: Trends and Priorities in Research and Delivery. *Proceedings of the 2nd International Conference on Rehabilitation Engineering - Ottawa*, 23-26.

Vanderheiden, G. C. (1984B). High and Low Technology Approaches in the Development of Communication Systems for Severely Physically Handicapped Persons. *Exceptional Education Quarterly*. (Vol. 4, No. 4), 40-56.

Vanderheyden, P. B., Pennington, C. A., Peischl, D. M., McKnitt, W. M., McCoy, K. F., Demasco, P. W., van Balkom, H., and Kamphuis, H. (1994). Developing AAC Systems that Model Intelligent Partner Interactions: Methodological Considerations. *Proceedings of the RESNA '94 Annual Conference*, M. Binion (ed.), RESNA Press, Arlington, VA, 126-128.

VanDyke, J. A. (1991). Word Prediction for Disabled Users: Applying Natural Language Processing to Enhance Communication. Thesis for Honors Bachelor of Arts in Cognitive Studies, University of Delaware, Newark, DE. 1991.

Velardi, P., Fasolo, M., and Pazienza, M. T. (1991). How to Encode Semantic Knowledge: A Method for Meaning Representation and Computer-Aided Acquisition. *Computational Linguistic* (Vol. 17, No. 2), 153-170.

Venkatagiri, H. S. (1993). Efficiency of Lexical Prediction as a Communication Acceleration Technique. *Augmentative and Alternative Communication* (Vol. 9, September), 161-167.

Vizetelly, F. H. (1915). *The Development of the Dictionary of the English Language*. Funk and Wagnalls publishers, New York.

von Tetzchner, S. (1988). Aided Communication for Handicapped Children. *Ergonomics in Rehabilitation*, Mital & Karwowski (eds.), Taylor and Francis Ltd. Publishers, 233-252.

Waller, A., Broumley, L., and Newell, A. F. (1992). Incorporating Conversational Narratives in an AAC Device. *Presented at ISAAC-92. Abstract appears in Augmentative and Alternative Communication*, 8.

Weischedel, R. M., and Sondheimer, N. K. (1983). Meta-Rules as a Basis for Processing Ill-Formed Input. *American Journal of Computational Linguistics* (Vol. 9, Nos. 3-4), 161-177.

Wilks, Y. (1975). An Intelligent Analyzer and Understander of English. *Communications of the ACM* (Vol.18, No.5), 264–274.

Winograd, T. (1983). *Language as a Cognitive Process. Volume 1: Syntax*. Addison-Wesley Publishing Company, Reading, MA.

Zernik, U. (1989). Lexicon Acquisition: Learning from Corpus by Capitalizing on Lexical Categories. *DARPA Speech and Natural Language Workshop*, February 1989, Philadelphia, 1556-1562.

Zickus, W. M. (1994). A Comparative Analysis of Beth Levin's English Verb Class Alternations and WordNet's Senses for the Verb Classes HIT, TOUCH, BREAK and CUT. *Proceedings of the Post-COLING94 International Workshop on Directions of Lexical Research*, Tsinghua University, Beijing, 66-74.

Zickus, W. M., McCoy, K. F., Demasco, P. W., and Pennington, C. A. (1995). A Lexical Database for Intelligent AAC Systems. *RESNA 95*. Vancouver. June 1995. To appear in 1995 proceedings.

Appendix A

LAD SOURCE DESCRIPTION

Base Class: lad

The **lad** class is a NULL base class. All the classes that will access different lexical and linguistic sources will be derived from the **lad** class. In this manner there will be a minimum of code changes necessary in order to increase LAD's usefulness. All of the functions are virtual functions will be overloaded in the derived classes. When new classes are added to LAD, they will be public derived from **lad**. If a new class has functions not already part of the LAD design, these new methods will need to be added to the list of virtual functions that are in the **lad** class, and their overloaded definitions will be in the derived class for which they are useful, and alterations will need to be made to `main.cc` (the driver for the LAD module) in order to enable access to the new class methods. Any matters concerning control will be handled either within the methods themselves or in the driver. Table A.1 lists the **lad** functions and specifies the allowable arguments for these functions. This table is followed by a description of the functions, and the actions they provide.

Table A.1: lad Class

Function Names	Function Arguments
lad()	
~lad()	
virtual valList* listCategories	char* word, char* typesearch
virtual int isWordRole	char* word, char* wordpos, char* typesearch, char* searchrole
virtual int isInSD	char* word
virtual verbFrame* getVerbFrame	char* word
virtual valList* getWNVerbFrame	char* word
virtual verbFrame* makeVerbFrame	char* word
virtual verbFrame* getVerbFrameSyn	char* word, int flag
virtual valList* getVerbSynonym	char* word
virtual valList* getAllVerbSynonym	char* word
virtual int isan	char* word, char* parent
virtual int isav	char* word, char* parent
virtual int hasa	char* word, char* part
virtual char* mappings	char* key

All functions are virtual in the base class **lad**, except for the constructor and destructor. See below for detailed descriptions:

`lad::lad()`
Since this is a NULL base class, this constructor does nothing.

`lad::~lad()`
This destructor destroys nothing, since this is a NULL base class.

`virtual valList* lad::listCategories(char* word, char* typesearch)`
This function takes as arguments **word**, the word being searched for, and a char* representing the type of search to be done, **typesearch**. This function returns 0 in the **lad** class, as it is further defined in the derived **wordnet** class.

`virtual int lad::isWordRole(char* word, char* wordpos, char* typesearch, char* searchrole)`
This function takes as arguments a **word** with its **wordpos** (NOUN, VERB, etc.), a **typesearch** to indicate the kind of search you are doing (*hypen/hypev* for an "isa" search, *partn* for a "hasa" search, and *coorn* for a "coordinate term" search) and **searchrole** (the role name we are looking to see if **word** has in its hierarchy). This function returns 0, as being in the null base class **lad**, it is further defined in the derived **wordnet** class.

`virtual int lad::isInSD(char* word)`
This function takes as an argument **word**, and checks to see if it is in the secondary database containing verb case frames. This function returns 0, as being in the null base class **lad**, it is further defined in the derived **SecDB** class.

`virtual verbFrame* lad::getVerbFrame(char* entry)`
This function takes an argument, **entry**, and returns 0, since it is further specified in the derived **SecDB** class.

`virtual valList* lad::getWNVerbFrame(char* entry)`
This function takes a verb and searches the WordNet VERB database for that verb. It returns 0, since it is in the null base class and is further defined in the derived **wordnet** class.

`virtual verbFrame* lad::makeVerbFrame(char* entry)`
This function takes a string (which is a WordNet VERB frame, see Appendix D) and returns 0. This function is defined in the derived **wordnet** class.

`virtual verbFrame* lad::getVerbFrameSyn(char* entry)`
This function takes a verb as its argument and returns 0, as it is further specified in the derived **wordnet** class.

`virtual valList* lad::getVerbSynonym(char* entry)`
This function takes a verb as its argument and returns 0, as it is further specified in the derived **wordnet** class.

`virtual valList* lad::getAllVerbSynonym(char* entry)`
This function takes a verb as its argument and returns 0, as it is further specified in the derived **wordnet** class.

`virtual int lad::isan(char* word, char* parent)`
This functions takes a word and its parent-term as arguments and returns 0 in this base class, as it is defined in the derived **wordnet** class.

`virtual int lad::isav(char* word, char* parent)`
This functions takes a word and its parent-term as arguments and returns 0 in this base class, as it is defined in the derived **wordnet** class.

`virtual int lad::hasa(char* word, char* part)`
This functions takes a word and its part as arguments and returns 0 in this base class, as it is defined in the derived **wordnet** class.

`virtual char* lad::mappings(char* key)`
This functions takes a word as its argument and returns 0 in this base class, as it is defined in the derived **wordnet** class.

Public Derived Class: wordnet : public lad class

The **wordnet** class handles all the interactions with the WordNet library functions and databases. Every function in this class extracts the information needed for the LAD methods that WordNet can provide (e.g., hierarchical, sense, synonym, and coordinate term data). This class is not complete, as there is more information that can be extracted from WordNet than is currently being done (see Chapter 3, Functional Interface section). A list of the functions implemented in the **wordnet** class is shown in Table A.2.

Table A.2: wordnet Class

Function Names	Function Arguments
wordnet()	
~wordnet()	
valList* listCategories	char* word, char* typesearch
int isWordRole	char* word, char* wordpos, char* type-search, char* searchrole
valList* getWNVerbFrame	char* word
verbFrame* makeVerbFrame	char* word
verbFrame* getVerbFrameSyn	char* word, int flag
valList* getVerbSynonym	char* word

Table A.2: wordnet Class

Function Names	Function Arguments
valList* getAllVerbSynonym	char* word
int isan	char* word, char* parentTerm
int isav	char* word, char* parentTerm
int hasa	char* word, char* part
char* mappings	char* key

The following are function descriptions of the **wordnet** class:

`wordnet::wordnet()`
Since this class has no private members, the constructor does nothing.

`wordnet::~wordnet()`
This destructor destroys nothing, since this class has no private members.

`valList* wordnet::listCategories(char* word, char* type-search)`
This function takes in a key, **word**, and a string representing the type of search to be done, **typesearch**, as its arguments. It searches the WordNet NOUN database for **word** and then checks its **isa** hierarchical links looking to see if **word** can fill the semantic categories used in our semantic reasoning. This function calls the **mappings** function to enable a mapping of the semantic categories used by the SRM and Compansion systems to the category names used by WordNet. It returns a list of the possible semantic categories **word** can play. For example: **listCategories(hammer, hypen)** returns the list **[object, inanimate, tool]**.

`int wordnet::isWordRole(char* word, char* wordpos, char* typesearch, char* searchrole)`
This function takes as arguments a **word**, its **wordpos** (NOUN, VERB, etc.), a **type-search**, to indicate the kind of search you are doing (*hypen/hypev* for an **isa** search, *partn* for a **hasa** search, and *coorn* for a **coordinate term** search) and the object of its **searchrole**. For example, **isWordRole(window, NOUN, hypen, object)** searches the WordNet NOUN database using the *hypen* pointer to determine if a *window* **isa** *object*. The output from this function is a integer, 0 for false and 1 for true. In the example shown, the output is a 1 for true.

`valList* wordnet::getWNVerbFrame(char* entry)`
This function takes a verb, **entry**, and searches the WordNet VERB database for that verb. It will locate the verb frames for that verb and then call the **makeVerbFrame** function to

convert the WordNet verb frame to an appropriate specified case frame. This will return a list of the possible specified case frames for **entry**.

`verbFrame* wordnet::makeVerbFrame(char* entry)`
This function takes a string (which is the WordNet verb frame, see Appendix D) and convert this into a specified case frame. It is called by **getWNVerbFrame** and returns a specified case frame for **entry**.

`verbFrame* wordnet::getVerbFrameSyn(char* entry)`
This function takes a verb, **entry**, that is not in the secondary database, and will go to WordNet searching for synonyms of **entry**. Then it will search the secondary database for a specified case frame using the list of synonym terms. It is call by **getVerbFrame** and will return a specified case frame or a 0 if no frames are found for any of the verb's synonyms.

`valList* wordnet::getVerbSynonym(char* entry)`
This function takes a verb, **entry**, and searches the WordNet VERB database for that verb. It returns a list of synonyms for **entry**, or it will return a 0. This function is called by the **getVerbFrameSyn** function.

`valList* wordnet::getAllVerbSynonym(char* entry)`
This function takes a verb, **entry**, and searches the WordNet VERB database for that verb. It returns a list of synonyms for that verb and that verb's superordinates or it will return a 0. This function is called by the **getVerbFrameSyn** function.

`int wordnet::isan(char* word, char* parent)`
This function takes a noun, **word**, and a parent-term, **parent**, as input. It searches WordNet's NOUN database for **word**, and then looks to see if the parent-term is in the hierarchy for **word**. It if finds **parent**, it returns a 1 for true, else it returns a 0 for false.

`int wordnet::isav(char* word, char* parent)`
This function takes a noun, **word**, and a parent-term, **parent**, as input. It searches WordNet's VERB database for **word**, and then looks to see if the parent-term is in the hierarchy for **word**. It if finds **parent**, it returns a 1 for true, else it returns a 0 for false.

`int wordnet::hasa(char* word, char* part)`
This function takes a noun, **word**, and a part-term, **part**, as input. It searches WordNet's NOUN database for **word**, and then looks to see if the part-term is in the attribute list for **word**. It if finds **part**, it returns a 1 for true, else it returns a 0 for false.

`char* wordnet::mappings(char* key)`
This function is called by the **listCategories** function, while it is parsing the hierarchical lexical link of a word. As each word of this link is parsed, it is passed as the argument to

the **mappings** function, so it can be compared to entries in the two-dimensional **translation** array (see Appendix C). If **key** is located in the first column of the array, the entry in the second column of **trans** at the same index is returned. In this manner, we are able to map the semantic categories used by the SRM or Compansion to the categories used by WordNet.

Public Derived Class: SecDB : public lad class

Presently the functions in this class are used to obtain verb case frame information. Table A.3 lists the functions implemented for this class and the valid arguments for these functions.

Table A.3: SecDB Class

Function Names	Function Arguments
SecDB()	
~SecDB()	
verbFrame* getVerbFrame	char* word
int isInSD	char* entry

The following are functions descriptions of the **SecDB** class:

`SecDB::SecDB()`
Since this class has no private members, the constructor does nothing.

`SecDB::~SecDB()`
This destructor destroys nothing, since this class has no private members.

`case_fr* SecDB::getVerbFrame(char* word)`
This function searches for a case frame for a given verb, **word** in the secondary database. The path of control for this search is to first check in the secondary database for **word** by calling the **isInSD** function; if it is there, then the associated case frame is returned. If no case frame is found, then it searches WordNet for synonyms of **word** by calling the **getVerbFrameSyn** function. This **wordnet** class function will traverse this list of synonyms (checking the secondary database for a synonymous entry); if it finds an entry, it will return the associated case frame. If no case frame is found, then it searches WordNet for

synonyms of the parent of **word** and again will check the secondary database for entries. If **getVerbFrameSyn** finds no case frame, then it searches WordNet for **word** and will generate a list of possible case frames from the verb frames by calling the **getWNVerbFrame** function. The output from this function is a list of case frames for **word**.

`int SecDB::isInLRD(char* entry)`
This function is used in the **getVerbFrame** function. It searches the secondary database to see if a word is entered there, and if it is it returns a 1, for true, otherwise it returns a 0, for false.

Appendix B

SRM SOURCE DESCRIPTION

Base Class: filledCaseFrame : public val class

The **filledCaseFrame** class defines the members and methods used to create filled case frames. These case frames are used by the SRM to provide the user with a list of frames that show all of the possible generations from the words of input. This class uses a struct, **f_case**, that contains a char* **caseName**, char* **caseFiller**, and an int **caseRating**, thus allowing the SRM to specify the role the case is filling (caseName), the word of input that will fill this role (caseFiller) and the rating given to the placement of that word into that case. (e.g., for the verb *break*, Mary would be given a rating of 12 for the role of agexp and a rating of 3 for filling either the goal or benef roles). Table B.1 provides a list of the private members of the **filledCaseFrame** class (e.g., the verb, total frame rating, and the eight **f_case** roles provided in every frame), the methods used to manipulate the data stored in these members, and the arguments that are valid for the methods.

The **filledCaseFrame** class is derived from the **val** class (a class implemented to handle integer and string input in various manners (e.g., in registers, lists and alone)), to utilize the class methods that enable making lists of **filledCaseFrame** objects. In this manner, we have profited from the object-oriented advantage of re-usable code.

Table B.1: filledCaseFrame Class

PRIVATE MEMBERS	FUNCTIONS	FUNCTION ARGUMENTS
char* caseVerb	filledCaseFrame	
int filledCaseRating	filledCaseFrame	char* name
f_case* agexp	~filledCaseFrame	
f_case* theme	int isSet	char* role
f_case* instr	int getType	
f_case* goal	f_case* findCase	char* role
f_case* benef	void setName	char* role, char* name
f_case* loc	void setRating	char* role, int rate
f_case* timee	void setFiller	char* role, char* entry
f_case* tense	void setVerb	char* newverb
	void setFilledCaseRating	
	void fillCases	char* role, int rate, char* cat, char* newword, char* newverb
	void print	ostream &os

The following function descriptions give detailed information about the arguments, actions, and output they provide.

`filledCaseFrame::filledCaseFrame()`
This constructor instantiates the **f_case*** members with 0 (NULL) for their **caseName**, **caseFiller** and **caseRating** items, the **caseVerb** to 0 (NULL) and the **filledCaseRating** to 0.

`filledCaseFrame::filledCaseFrame(char* name)`
This constructor instantiates the **f_case*** members with 0 (NULL) for their **caseName**, **caseFiller** and **caseRating** items, the **caseVerb** to **name** and the **filledCaseRating** to 0.

`filledCaseFrame::~filledCaseFrame()`
This destructor deletes the **f_case*** members and returns the **caseVerb** and **filledCaseRating** to 0 (NULL).

`int filledCaseFrame::isSet(char* role)`
This function finds the specific case designated by **role**, and returns a 1 if it has already been filled or else it returns a 0. This function is used to ensure that once a case role has been filled by a word, that another word does not overwrite the data. This function is called by the **setName**, **setRating** and **setFiller** methods.

`int filledCaseFrame::getType()`
This function is inherited from the **val** class. It is used internally by other functions to enable the system to get type information from an object.

`f_case* filledCaseFrame::findCase(char* role)`
This function returns a **f_case*** pointer to the case specified by **role**. This function is used by **setName**, **setRating**, and **setFiller** to locate the proper case before filling it with the proper data.

`void filledCaseFrame::setName(char* role, char* name)`
This function gets a pointer to the proper case by calling **findCase(role)**, then initializes this case's **caseName** with **name**.

`void filledCaseFrame::setRating(char* role, int rating)`
This function gets a pointer to the proper case by calling **findCase(role)**, then initializes this case's **caseRating** with **rating**.

`void filledCaseFrame::setFiller(char* role, char* entry)`
This function gets a pointer to the proper case by calling **findCase(role)**, then initializes this case's **caseFiller** with **entry**.

`void filledCaseFrame::setFilledCaseRating()`
This function totals up the **caseRating** integers for all of the cases in the filled case frame and then initializes the **filledCaseRating** member with this total.

`void filledCaseFrame::setVerb(char* newverb)`
This function initializes the **caseVerb** member of the filled case frame with **newverb**.

`void filledCaseFrame::fillCases(char* role, int rate, char* cat, char* newword, char* newverb)`
This function calls **setVerb(newverb)**, **setRating(role, rate)**, **setFiller(role, cat)**, and **setName(role, newword)** to initialize a case within a filled case frame.

`void filledCaseFrame::print(ostream &os)`
This function prints out the contents of a filled case frame.

base class: verbFrame

The **verbFrame** class contains the members and methods needed to represent and manipulate specified case frames. The individual cases within the frame are made up of **slot*** structures which contain a string **caseName**, to indicate the role name associated with that **slot*** case, an integer **toFillPref**, to provide the semantic preference placed on that **slot*** case, and a valList* **fillWith**, to provide the list of semantic categories that can fill that case and the ratings associated with each category on this list.

The specified case frames are the backbone of the semantic reasoning for the SRM. They provide the semantic preference information, as well as details as to what kinds of words can fill a specific role in a sentence. There are twenty different types of verb specified case frames that are depicted by the twenty derived classes from the **verbFrame** class. They differ in the **fillWith** and **toFillPref** members of their **slot*** roles, and provide semantic reasoning for a variety of different kinds of verbs. These twenty derived classes are: relational, attributive, verbal, written, oral, material, erg_do (ergative do verbs), ani_do (animate do verbs), peop_do (people do verbs), ingest, drink, eat, inhale, PA_trans, mental, cognitive, sensory, tactile, visual and auditory. Their details are not included in this Appendix since they differ from the **verbFrame** class in the contents of the two previ-

ously mentioned **slot*** members, but inherit their functions from the **verbFrame** class. Table B.2 contains a list of the private members, the functions, and the valid arguments which make up the **verbFrame** class.

Table B.2: verbFrame Class

PRIVATE MEMBERS	FUNCTIONS	FUNCTION ARGUMENTS
char* caseVerb	verbFrame	
slot* agexp	verbFrame	char* name
slot* theme	~verbFrame	
slot* instr	valList* genFilledFrames	valList* values, char* verb, valList* mand, valList* used
slot* goal	filledCaseFrame* filledFrames	valList* mand, char* verb
slot* benef	valList* genMultFrames	valList* tmp2list, valList* mandroles, char* verb
slot* loc	int complexList	valList* mult, valList* role, valList* mandRoles, char* verb, valList* retlist, filledCaseFrame* frame, int recurFlag, int wdct, int recurctr
slot* timee	void removeCat	valList* rolelist
slot* tense	void ridConflicts	valList* used, valList* checking
	slot* get_frame	char* role
	int isType	
	void set_role	char* entry
	char* get_role	char* role
	void setToFillPref	char* role, int pref
	void setFillWithPref	char* role, valList* list
	int getToFillPref	char* role

Table B.2: verbFrame Class

PRIVATE MEMBERS	FUNCTIONS	FUNCTION ARGUMENTS
	valList* getFillWithPref	char* role
	int getRegIntValue	valReg* item
	int getCatRating	char* role, char* category
	int numHighRating	char* role
	int getMaxRating	char* role
	valList* findAllRoles	char* word
	valList* findBestRoleNRate	char* word
	int findBestRate	char* word
	valList* findBestRole	char* word
	valList* getMaxCatStr	char* role
	int getMaxFillWithPref	char* role, char* word
	char* getStrAssocMaxPref	char* role, char* word
	void print_frame	

The following descriptions give detailed information about the arguments, actions, and output the functions listed in Table B.2 provide:

`verbFrame::verbFrame()`
This constructor instantiates the eight **slot*** members with 0 (NULL) for their **toFillPref**, and **fillWith** items, their **slot*** member **caseName** to "agexp", "theme", "instr", "goal", "benef", "loc", "timee", and "tense" (respectively), and the **caseVerb** to 0 (NULL).

`verbFrame::verbFrame(char* name)`
This constructor instantiates the eight **slot*** members with 0 (NULL) for their **toFillPref**, and **fillWith** items, their **slot*** member **caseName** to "agexp", "theme", "instr", "goal", "benef", "loc", "timee", and "tense" (respectively), and the **caseVerb** to **name**.

verbFrame::~verbFrame()
This destructor deletes the **slot*** members and returns the **caseVerb** to 0 (NULL).

valList* verbFrame::genFilledFrames(valList* values, char* verb, valList* mand, valList* used)
This function is responsible for generating the **filledCaseFrame*** objects that are the final output of the SRM. The algorithm it follows is: 1) check for all words that can only fill one role from the **values** list and put them on the **mand** list; 2) call **ridConflicts** to eliminate the used roles from Step 1 out of the remaining word's rolelists; 3) go back to step 1 until there are no conflicting roles left on the rolelists of the remaining words; and 4) if there are no words left with multiple roles they can fill, then all the words are on the **mandroles** list, so call **filledFrames** to generate the **filledCaseFrames*** from the **mandroles** list, else call **genMultFrames** to handle the recursion required for further processing.

filledCaseFrame* verbFrame::filledFrames(valList* mand, char* verb)
This function takes a list of words that can only fill one case and generates and returns the **filledCaseFrame*** for this list.

valList* verbFrame::genMultFrames(valList* tmp2list, valList* mandroles, char* verb)
This function takes a list of words that can play multiple roles with their lists of roles, a list of roles that are already filled because one or more words can only fill one role, and the **verb** of input as arguments. It checks to determine if **tmp2list** is a one element list, if it is then the function simply generates filled frames for each role in the one element list making sure there are no conflicts (i.e., checking to ensure that these roles are not already filled by the mandroles list elements) and adds the categories that are in the **mandroles** list. If **tmp2list** is more than one element, then more complex reasoning needs to be done so the function calls **complexList** to handle the intricacies of finding all combinations of all of the roles on all of the lists that are on **tmp2list** without any conflicting roles occurring.

int verbFrame::complexList(valList* mult, valList* role, valList* mandRoles, char* verb, valList* retList, filledCaseFrame* frame, int recurFlag, int wdct, int recurctr)
This function has the job of finding all combinations of cases a word can play in a filled case frame when there is a multiple list of words, **mult**, that can each fill multiple cases, making sure there are no words filling the same case (this is a case conflict) and that all the words are used in every filled frame, while ensuring that the cases on **mandRoles** are not filled since they are already filled by words that can only fill one role and thus must fill these roles. This function is called recursively, this means that it is called within the

processing of a call to this function, which is why the variables **recurFlag**, **wdct**, and **recurctr** are required. The final output, **retList**, is a list of filledCaseFrame objects.

`void verbFrame::removeCat(valList* rolelist)`
This function will remove a category from a list, **rolelist**. This is used in the **ridConflicts** function to remove categories from lists if they are mandatorily filled by words that can only fill one role.

`void verbFrame::ridConflicts(valList* used, valList* checking)`
This function searches the **checking** list for rolelists with entries that match entries of the **used** list. If matches are found, they are removed from the rolelists. The **used** list is a list of roles that must be maintained since they are the only possible role their associated word can fill.

`slot* verbFrame::get_frame(char* role)`
This function returns a **slot*** pointer to the case corresponding to **role** (e.g., **get_frame(agexp)** will return a pointer to the **slot*** agexp).

`int verbFrame::isType()`
This function returns the integer value associated with VERB, it is used to find type of a **verbFrame*** object. This is needed because there are twenty types of specified verb frames derived from this class, each one having its own enumerated value.

`void verbFrame::set_role(char* entry)`
This function sets the **caseVerb** to **entry**.

`char* verbFrame::get_role(char* role)`
This function returns the **caseName** of the case corresponding to **role**.

`void verbFrame::setToFillPref(char* role, int pref)`
This function locates the case associated with **role** and sets the **toFillPref** with **pref**. For instance, **setToFillPref("agexp", 4)** will find the **slot*** case for agexp and set its **toFillPref** to 4.

`void verbFrame::setFillWithPref(char* role, valList* list)`
This function locates the **slot*** case corresponding with **role** and sets it **fillWith** to **list**. For instance **setFillWithPref("theme", category_list)** will find the **slot*** case for theme and initialize its **fillWith** value to **category_list**.

`int verbFrame::getToFillPref(char* role)`
This function locates the **slot*** case that corresponds to **role** and returns its **toFillPref** integer value.

`valList* verbFrame::getFillWithPref(char* role)`
This function locates the **slot*** case that corresponds to **role** and returns its **fillWith valList*** value.

`int verbFrame::getRegIntValue(valReg* item)`
This function extracts the integer value from a **valReg*** item. There are functions in the **valReg*** class that extract the register contents, but as a **val*** item and for this class, we need the value returned as an integer, not a **valInt*** item.

`int verbFrame::getCatRating(char* role, char* category)`
This function locates the **slot*** case corresponding to **role** and searches the **fillWith** list for an entry that matches **category**. Then this method returns the integer value associated with **category**, if **category** was found, otherwise it returns 0.

`int verbFrame::numHighRating(char* role)`
This function locates the **slot*** case corresponding to **role** and searches the **fillWith** list for the all entries with rating equal to the maximum rating found from the function call to **getMaxRating** and returns the total number of these entries found.

`int verbFrame::getMaxRating(char* role)`
This function locates the **slot*** case corresponding to **role** and searches the **fillWith** list for the entry with the highest rating and returns this maximum rating.

`valList* verbFrame::findAllroles(char* word)`
This function searches WordNet NOUN database for **word**. The goal of this function is to locate all of the semantic categories **word** has that matches the semantic category roles that are in the **fillWith** lists associated with the eight cases for the **verbFrame*** object that calls this function, and to place these in the **valList* return_list** and then return this list. It needs to ensure that a specific semantic category is not added to the **return_list** more than

once (since this category could be in more than one of the eight **fillWith** lists). The **return_list** is a list of **valReg*** registers containing the semantic category name and the integer rating associated with that name.

`valList* verbFrame::findBestRoleNRate(char* word)`
This function has not been implemented, it is intended to find the best role and rate for a word.

`int verbFrame::findBestRate(char* word)`
This function calls **findAllRoles** to find all the roles **word** can fill and the associated integer ratings with these different roles, and then returns the maximum of these ratings.

`valList* verbFrame::findBestRole(char* word)`
This function calls **findBestRate** to determine the maximum rating **word** can provide and then locates all the roles **word** can fill that have this rating and places these roles on a list. The output from this function is a list of maximum rated roles a word can fill in a case frame.

`valList* verbFrame::getMaxCatStr(char* role)`
This function calls **getMaxRating(role)** and **numHighRating(role)** in order to return a list of categories that have the same maximum rating for a specific case in a verb frame.

`int verbFrame::getMaxFillWithPref(char* role, char* word)`
This function calls **getMaxRating(role)** for a specific case corresponding to **role** and returns this maximum integer value.

`char* verbFrame::getStrAssocMaxPref(char* role, char* word)`
This function has not been implemented, it is intended to return the word associated with the maximum rating.

`void verbFrame::print_frame()`
This function prints out the contents of a **verbFrame*** specified case frame.

Appendix C

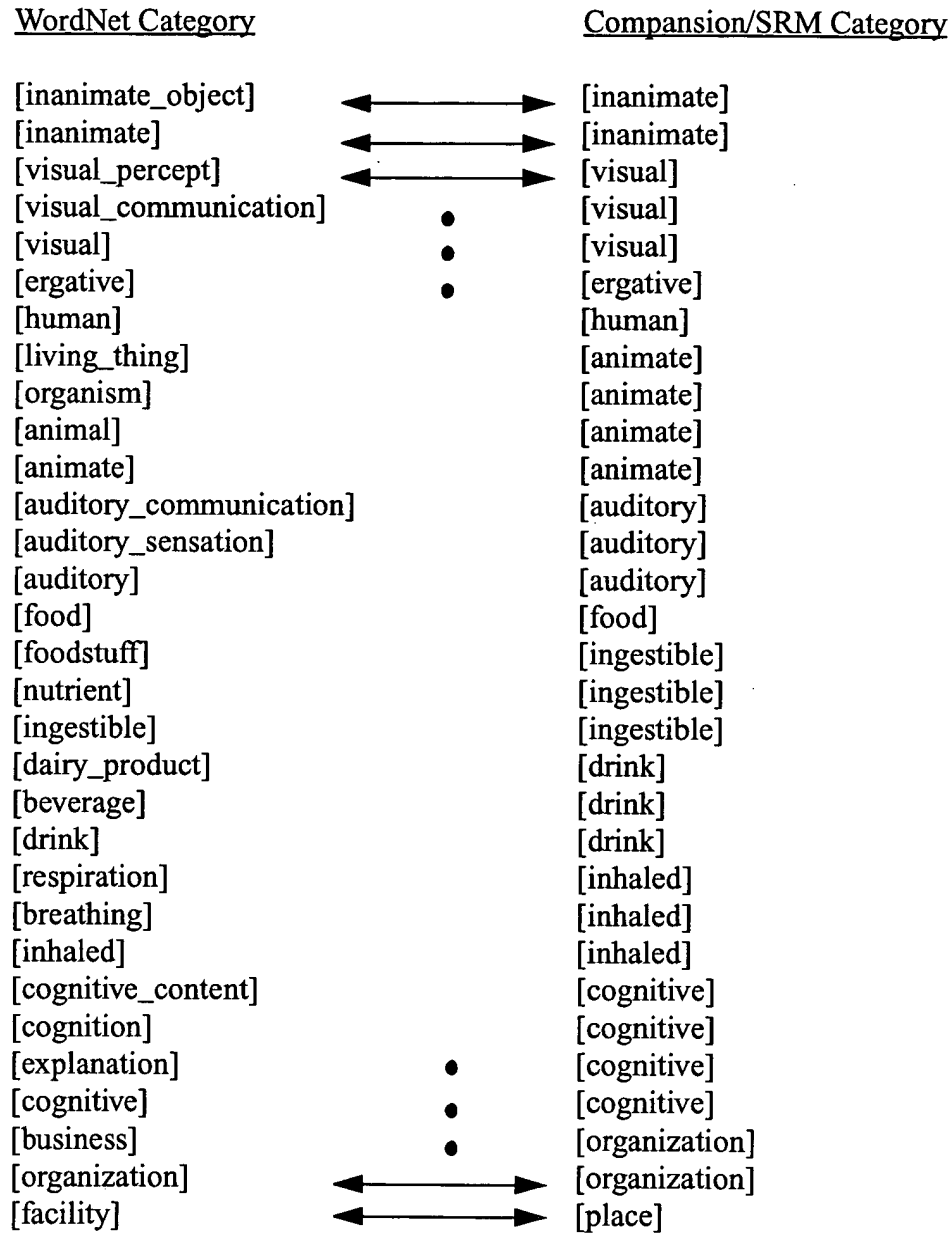
SEMANTIC CATEGORIES

One of the major projects of the ASEL NLP lab, Compansion, has its verb hierarchy based on a systemic grammar that utilizes semantic categories to capture semantic preference ratings. Since it is envisioned that LAD will be interacting with several various lexical and linguistic resources, there is a need to chose a base of semantic categories that can then be mapped to the semantic categories of these multiple resources. We chose to use the semantic categories used in the current implementation of Compansion for the SRM as well.

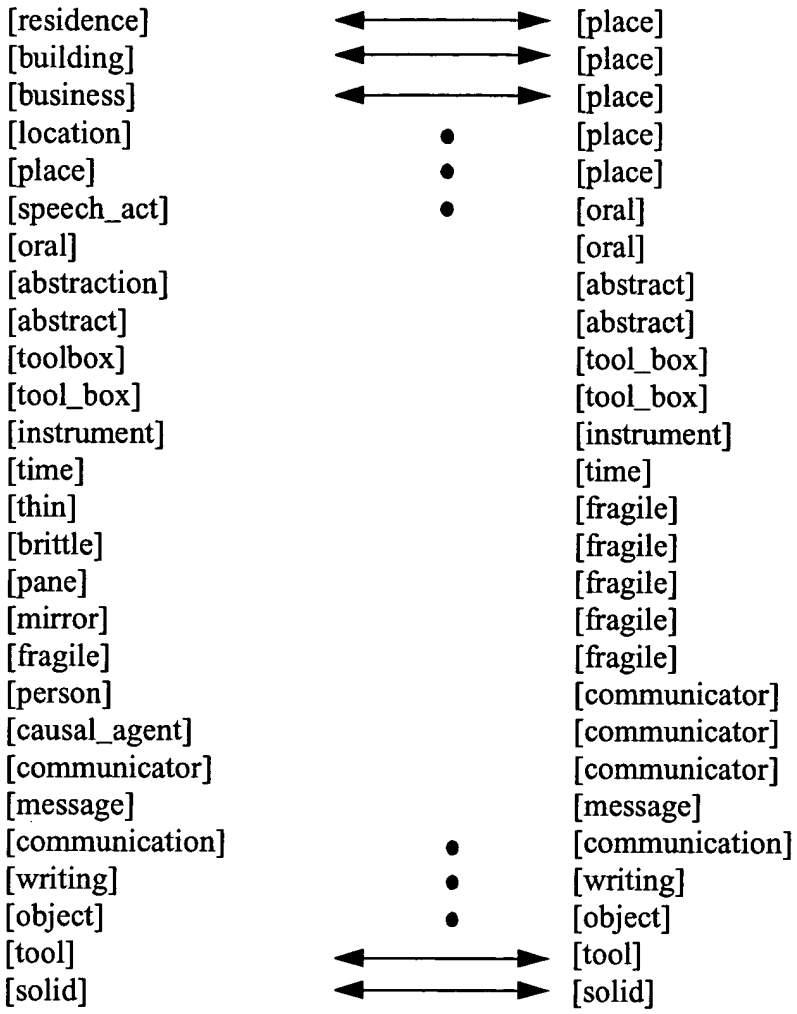
Below is the current list of semantic categories we use:

animate
inanimate
instrument
organization
fragile
human
physical
place
time
communicator
message
abstract
communication
writing
oral
object
ergative
tool_box
tool
solid
ingestible
food
description

In order for LAD to search the WordNet databases for these semantic categories, a mapping function was needed to map the above category names to the category names used in WordNet. To help facilitate this, the translation array was constructed as shown below:



(WordNet->Companion/SRM mapping continued)



Appendix D

WORDNET VERB FRAMES[1]

Something ----s
Somebody ----s
It is ---ing
Something is ---ing PP
Something ---s something Adjective/Noun
Something ---s Adjective/Noun
Somebody ---s Adjective
Somebody ---s something
Somebody ---s somebody
Something ---s somebody
Something ---s something
Something ---s to somebody
Somebody ---s on something
Somebody ---s somebody something
Somebody ---s something to somebody
Somebody ---s something from somebody
Somebody ---s somebody with something
Somebody ---s somebody of something
Somebody ---s something on somebody
Somebody ---s somebody PP
Somebody ---s something PP
Somebody ---s PP
Somebody's (body part) ---s
Somebody ---s somebody to INFINITIVE
Somebody ---s somebody INFINITIVE
Somebody ---s that CLAUSE

1. These WordNet verb sentence frames are from Fellbaum (1993).

Somebody ---s to somebody
Somebody ---s to INFINITIVE
Somebody ---s whether INFINITIVE
Somebody ---s somebody into V-ing something
Somebody ---s INFINITIVE
Somebody ---s VERB-ing
Somebody ---s something with something
It ---s that CLAUSE
Something ---s INFINITIVE